POLITECNICO DI TORINO

MASTER's Degree in MECHATRONIC ENGINEERING

MASTER's Degree Thesis

Pose classification for assistive unmanned vehicles with deep learning at the edge

Supervisors

Prof. MARCELLO CHIABERGE

VITTORIO MAZZIA

FRANCESCO SALVETTI

Candidate

DIEGO GIBELLO FOGLIO

Academic Year 2020-2021

"A te, che vegli constantemente su di me
e a voi, che mi avete insegnato ad essere
la persona che sono e sto diventando..."

Diego

Abstract

Technology in the last decades has revolutionized our way of life, making more and more powerful devices available to everyone. The development of such technologies has been exploited also to conduct researches in several scientific fields, i.e. medical area, biotechnologies, computer science etc. Nowadays, thanks to the improvement made in Robotics and Artificial Intelligence, there can be found virtual assistance devices in many households and public places.

Service robotics and artificial intelligence allows to help elder and disabled people, on different daily and basic tasks, in a noninvasive manner. The idea of this thesis was born at PIC4SeR (PoliTO Interdepartmental Center for Service Robotics) with the purpose of implementing a complex service robotics application to support elder and disabled people in the house environment.

The goal of this project is the development of an algorithm able to detect the pose of a person and recognize the static action done. The actions that can be recognized, in this first evolution of the work, are limited to three: standing, sitting and lying. Different data sets have been created to build several neural network architectures. These architectures have been tested to find the best trade-off between accuracy and computational cost to satisfy a real-time result, which is one of the main requirements of the system. The data sets have been created using different cameras, the pose estimation and action prediction are done using a stereo-camera, and an edge TPU coprocessor.

At the end of the experimentation, good results have been achieved giving an accurate prediction of the person's pose in the house environment.

Table of Contents

List of Figures

Acronyms

AI

Artificial Intelligence

ML

Machine Learning

NN

Neural Network

CNN

Convolutional Neural Network

RNN

Recurrent Neural Network

MLP

Multi Layer Perceptron

TPU

Tensor Processing Unit

ANN

Artificial Neural Network

PCA

Principal Component Analysis

API

Application Programming Interface

Chapter 1

Introduction

1.1 Objective

Technology in the last decades has revolutionized our way of life, making more and more powerful devices available to everyone. The development of such technologies has been exploited also to conduct researches in several scientific fields, i.e. medical area, biotechnologies, computer science etc. Nowadays, thanks to the improvements made in Robotics and Artificial Intelligence, there can be found virtual assistance devices in many households and public places.

Service robotics and artificial intelligence allows to help elder and disabled people, on different daily and basic tasks, in a noninvasive manner. The idea of this thesis was born at PIC4SeR (PoliTO Interdepartmental Center for Service Robotics) with the purpose of implementing a complex service robotics application to support elder and disabled people in the house environment.

The goal of this project is the development of an algorithm able to detect the pose of a person and recognize the static action done. The actions that can be recognized, in this first version of the work, are limited to three: standing, sitting and lying.
The developed algorithm is able to detect a static pose and not an action. The main difference is that an action recognition algorithm can detect a movement, a series of poses evolving in time. A batch of images are acquired and then a recognition algorithm is run. This project, instead, acquires and processes a single image at a time, which lead to a static pose recognition.

Different datasets have been created to build several neural network architectures. The models used to build the neural networks are two, Multi Layer Perceptron and Convolutional Neural Network. These architectures have been tested to find the best trade-off between accuracy and computational cost to satisfy a real-time result, which is one of the main requirements of the system. To do that an empirical approach is used testing the models varying only some parameter as the number of neurons present in each layer. To compare the models performances are taken into accounts four elements: the Loss and Accuracy behaviour of the Train and Validation set over the training epochs; the final Loss and Accuracy value; the Confusion Matrix; the behaviour of the Real time Pose. On the basis of these assumptions the three model are built.

The datasets have been created using different cameras. The first dataset, that is constituted by few images, is created using a photo camera, specifically a Canon 77D with a wide-angle lens. The successive datasets have been created using the same camera employed in the real time pose classification process, an Intel Realsense Depth Camera D435i. This camera allows to acquire also the depth map of the image to be able to retrieve the depth information about the human keypoints.
The pose estimation and action prediction are done using a stereo-camera, and an edge TPU coprocessor.

At the end of the experimentation, good results have been achieved giving an accurate prediction of the person's pose in the house environment.

1.2 Organization of the thesis

This thesis is structured in five chapters, whose content is presented below.

In *Chapter 1 - Introduction*, are presented the main objectives of the thesis, with a brief overview of its general structure. It is also presented an investigation into the concept of pose estimation and an analysis of some techniques employed in some recent projects about pose recognition and fall detection.

Chapter 2 - Machine Learning and Neural Networks offers a recap of the most important concepts of Machine Learning and Neural Networks, as Artificial Neural Network and Convolutional Neural Networks, to better understand the techniques used to develop the Pose Recognition algorithm.

Chapter 3 - Methodology, firstly, presents a detailed description about the pipeline used to develop the algorithm. Then, the cost function, the optimizer algorithm and the metrics used in the project are analyzed. Finally, is presented an overview of the hardware and software components.

In *Chapter 4 - Experimentation*, the creation of the different datasets and the architecture of the models is presented. First of all is described the image acquisition and the dataset creation using the two cameras. Then, is presented the pre-processing step. Finally the neural network architecture of the two models, MLP and CNN, is analyzed.

In *Chapter 5 - Results and Conclusions* all the obtained results are shown through tables and plots, finally including conclusions and suggestions on future work.

1.3 Related works

The goal of this project is the developing of an algorithm able to detect the pose of a person and recognize the static action done. The actions that can be recognized are standing, sitting and lying. It is a pose prediction and not an action recognition because of its working principle. As said in the survey "Human Action Recognition and Prediction: A Survey":

> The term *human action* studied in computer vision research ranges from the simple limb movement to joint complex movement of multiple limbs and the human body. This process is dynamic, and thus is usually conveyed in a video lasting a few seconds.[1]

The action recognition is a *dynamic* process and the algorithm, to recognize it, need to evaluate a batch of consecutive poses.
The algorithm developed for this project has been designed to analyze one image at a time, recognize seventeen body keypoints and, on the basis of them relative position, predict a pose. It is a sort of static action recognition process.

In this section various projects, concerning the pose and action recognition of single or multiple person and fall detection systems, are analyzed since have been a starting point for the developing of the thesis project.

1.3.1 Pose recognition

OpenPose

OpenPose is the fist real-time multi person system which is able to recognize a total number of 135 keypoints on a single image. These keypoints are divided into human body, foot, hand and facial points. During the iteration with objects, OpenPose provide robust results even if there are occlusions between the person and the object.
Figure 1.1 shows the real-time output of OpenPose considering all the recognizable features.

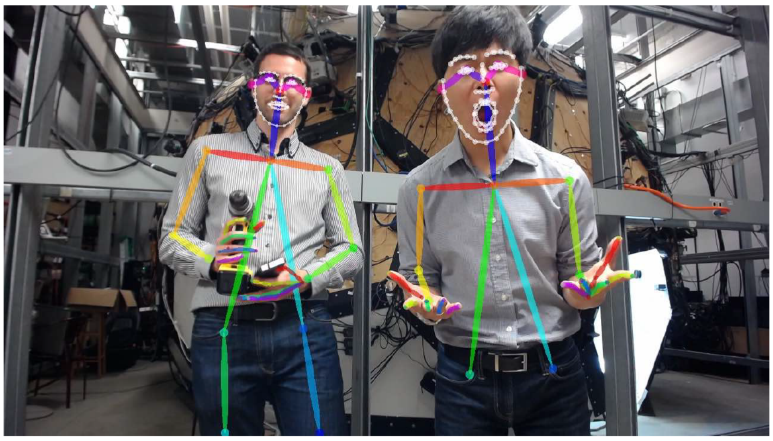


Figure 1.1: OpenPose output. [2]

To use OpenPose is not required the implementation of a pipeline, a frame reader, a display to visualize the output of the network, etc. OpenPose solves these problems providing a system that can be run over different platforms as Ubuntu, Windows, Mac OSX and embedded system. It is also able to provide support for GPU and only CPU devices.
The input of the network can be selected between images, webcam, video and IP camera streaming. Each detector (body, face, foot, hand) can be activated or not and the results can be displayed or saved on a memory peripheral.
As said by the developers: "OpenPose consists of three different blocks: body+foot detection, hand detection, and face detection. The core block is the combined body+foot keypoint detector".[2]
This algorithm is quite used for robotics application as person identification and other recognition techniques. Due to its popularity, the OpenCV library has included OpenPose within the Deep Neural Network module.

OpenPose has been the first choice for the implementation of the pose recognition algorithm. Due to its computational cost and the requirement of a GPU to achieve good real time results, it is substituted by a more simple algorithm PoseNet.

Coral PoseNet

PoseNet is an algorithm able to estimate the pose of a single person or multiple people.
Pose estimation is a computer vision method which allows to estimate the pose of a person from an image, a video or a real time camera, finding the position in space of the most relevant body joints, called also *keypoints*.
The Pose estimation process can be summarized in two phases:

1 The input RGB image is fed through a CNN, precisely a MobileNet V1 architecture. The algorithm processes the image producing a set of heatmaps (one for each kind of key point) and some offset maps. This phase runs on the Edge TPU device. The obtained results are fed to the second step.

2 An algorithm decodes the poses, the pose confidence scores, the keypoint positions, and the keypoint confidence scores. When the network is called, using the Coral Python API, a series of keypoints from the network is returned.

The object returned by PoseNet contains a list of keypoints and a confidence score for each person which has been detected.
The keypoints are seventeen relevant body parts, such as joints, eyes, nose, etc.
The complete scheme of the keypoints that PoseNet is able to detect is illustrated in the following figure:

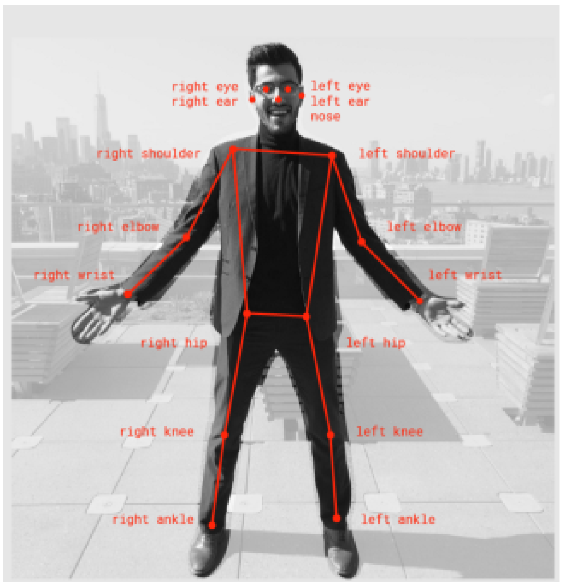


Figure 1.2: Coral PoseNet detected keypoints.[3]

Each keypoint contains two important information, as said by the Coral PoseNet developers. "The keypoint *confidence score* determines the confidence that an estimated keypoint position is accurate. It ranges between 0.0 and 1.0. It can be used to hide keypoints that are not deemed strong enough.
The keypoint *Position* is a 2D x and y coordinates in the original input image where a keypoint has been detected."[3]

Coral PoseNet has been used to develop the pose classification algorithm since, firstly an Edge TPU device has been used to speed up the computations, and secondly, it provides good results in terms of accuracy.

1.3.2 Action recognition

An important innovation for the computer vision field is the video analysis for human detection, identification and tracking. This analysis can lead to the development of systems that can detect the fall of a person or simply monitor an elder or disabled person in the house environment.

The human detection can be handled using a method called *Background subtraction*. The background subtraction separates the foreground of the image from its background. This algorithm has to satisfy some conditions to provide a valid output. It must react in a fast way to background changes, it must avoid the detection of non stationary objects and must be robust in the presence of illumination changes. As treated in [4] a background subtraction algorithm using *Double Frame Differencing* method [5] can be utilized to get the motion regions from a video frame. The obtained a algorithm has low computational cost and avoids the subtraction when no motion is detected.
This method is based on the AND operation between two images. Three consecutive images are acquired, the difference between the first frame and the second is computed repeating the operation also for the second and third frame. The AND operation is computed between the two resulting images obtaining a double difference images. The last step is the conversion of the image into a binary one. The background pixels are transformed into 0 and the foreground into 1. The last steps are the reduction of noise in the image and the detection of a bounding box around the moving person.

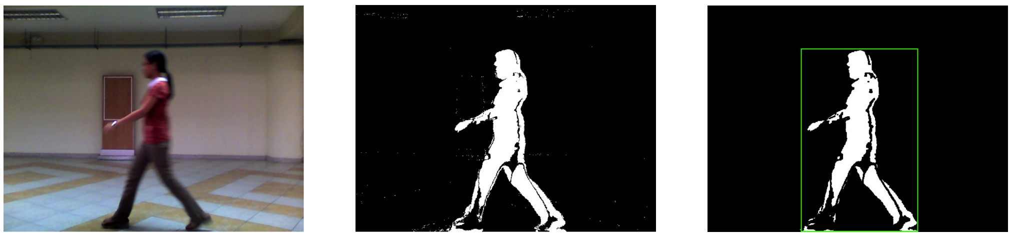

(a) Original frame **(b)** Processed frame **(c)** Bounding box

Figure 1.3: Background subtraction using Double Differencing algorithm and bounding box detection. [4]

The human pose recognition is another aspect of the computer vision field and can be handle in different ways.
The recent OpenPose and PoseNet algorithms have been previously analyzed, but

another pose estimation method can be examined.
This method developed in [4] is based on the extraction of feature using the projection of histograms. The extracted features are then used to train a neural network that outputs a model used to classify the pose.
Two histograms are derived from the binary image frame, an horizontal and a vertical histogram. These histograms are obtained from the foreground pixels, analysing the row and column wise components. For each pose (standing, sitting and lying) there are two histograms, one vertical and one horizontal. A vector, containing the feature extrapolated from the histograms, is formed and the training and classification steps can be developed.
The training of the model is done using *Support Vector Machines* (SVMs). SVMs are supervised learning methods used for classification and regression purposes. This algorithm classifies the data points finding an hyperplane in a X-dimensional space where X is the numbers of features. The researched hyperplane is the one which has the maximum distance between the points of the two classes. This reflects into a more robust classification. The obtained model is then used to derive a prediction vector than is fed to a classifier algorithm. This algorithm using some rules determines if the vector corresponds to one of the given poses.
The performances achieved by the whole system are not good since a very small database of images are used to test the system and the recognition rate using real time acquisition is not high (70%).

Despite the results obtained in [4], the pipeline and the methods which are used have been a starting point for the development of this thesis project.

Human detection, identification and tracking are quite simple process but are the basis for more complex projects like the fallen detection of an elderly person. To better understand how to treat the lying pose recognition, the paper [6] has been analyzed.
The innovation of [6] is the estimation of 3D human body keypoints starting from the 2D human pose and some depth information. The ground plane is calculated and, due to several measures, the system detects if the person is fallen or not. Another important aspect is that a Stereo Camera is used and the system is developed to be mounted on an autonomous robot.
The first step is the estimation of the 2D human pose that has been done using the approach of [7]. In [7] to detect the 2D pose is used a non parametric representation, called *Part Affinity Field*. Part Affinity Field is a feature representation method which in a 2D vector encodes the information about the orientation and the location of each limb. Each type of limb has a corresponding affinity field joining its two associated body parts.[7]
Once the 2D pose has been obtained it is calculated a confidence level for each person detected. If the confidence is above the 60% the 3D pose is calculated, otherwise the pose is discarded. The 3D pose is obtained using the depth images

and the camera matrix.
To proceed with the reasoning section the ground plane detection is needed. The ground plane detection has to satisfy three requirements: fast processing time, high reliability and slope tolerance. The detection is not done using a LIDAR but only a stereo camera.
The reasoning process is the core of this projects. Its inputs are the 3D ground plane and the position and confidence of each of the 3D detected points. To elaborate if a person is fallen or not [6] uses five metrics.

- Λ - It refers to the fallen detection confidence level threshold. If the detected human pose has a confidence value below Λ it is considered as false detection.
- **BSA** - It stands for Body Surface Area. It is used to identify a false detection. A bounding box around the detected person is estimated and the corresponding BSA is calculated. A false detection is considered if the BSA is below $1m^2$.
- **CoG** - It stands for Center of Gravity and it is calculated considering all the detected keypoints.
- **UbC** - It stands for Upper Body Critical. It is the center of gravity of a subset of the keypoints, in particular nose, eyes, ears, neck and shoulders are considered.
- **Ground Distance** - It merges 3D ground plane, CoG and UbC. It compares the points of each of the metrics calculating the euclidean distance. If one of the resulting distance is below $0.7m$ the detected pose is considered as a fallen detection.

With the reasoning phase all the system has been analyzed and the core steps are represented in the figure below (1.4)

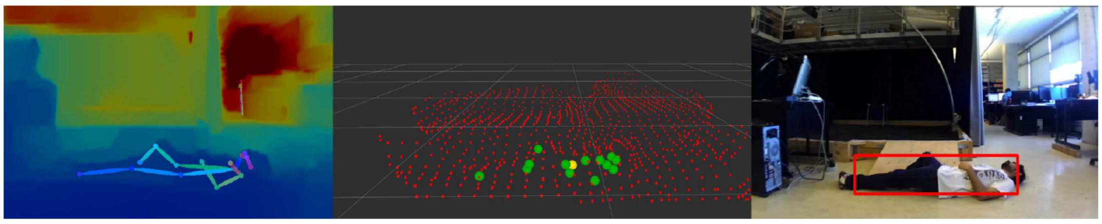


Figure 1.4: Main steps representation. The **left** image refers to the depth image with the detection of the 2D pose. In the **center** image are visualized the ground plane (red), the CoG (yellow) and the 3D keypoints (green). The **right** images shows the detection of a fallen person. [6]

The fallen person detector has been tested in a home and office environment giving as result an accuracy above the 91%. On of the important feature of the system is the ability to recognize in a correct way some poses with partially occluded body part or with few detected keypoints. This feature it is effective in a home environment where a lot of objects, especially when a person is lying on the ground, can occlude a body part.
Figure 1.5 shows the results achieved in the home environment and, as can be seen, it gives very good results even in different scenarios.

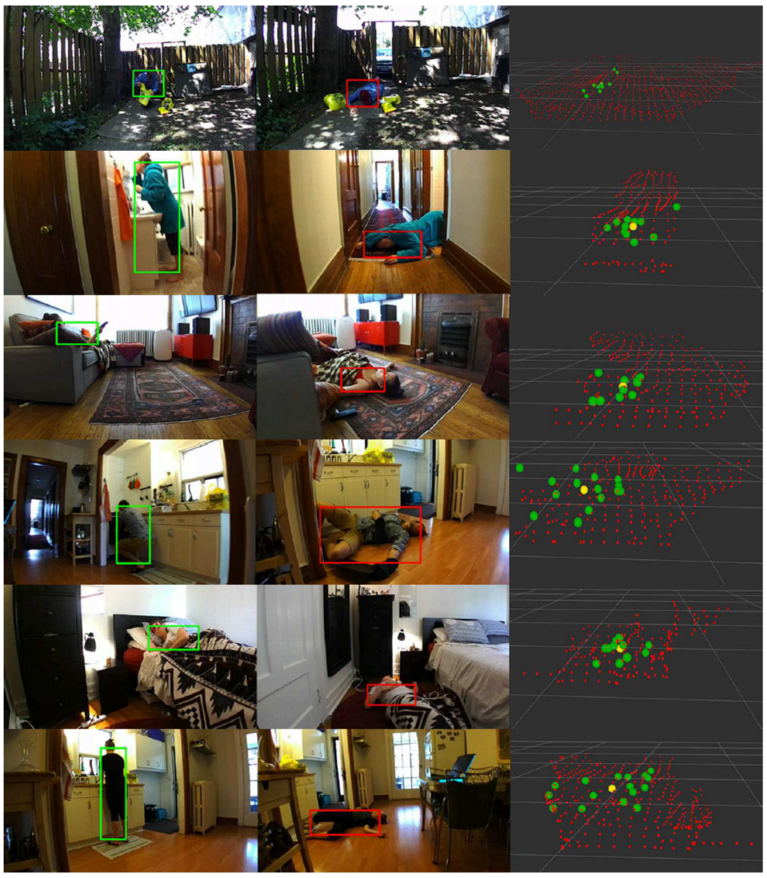


Figure 1.5: Right column: 3D keypoints and ground plane. Left column: scene with no fall detection. Right column: scene with fall detection. [6]

The arguments presented in this paper can be taken into account in a future evolution of the work to make a distinction between the lying pose and the detection of an accidental fall. An aspects that has been used in the realization of the thesis project is the utilization of the depth to generate 3D points to obtain better results in the pose recognition.

Chapter 2

Machine Learning and Neural Networks

2.1 What is and Why use Machine Learning

Machine Learning is the discipline of programming computers so they can learn from data and improve automatically though experience. This is a complete definition of Machine Learning but during the years several people gave their one. Arthur Samuel in 1959 gave a more general definition:

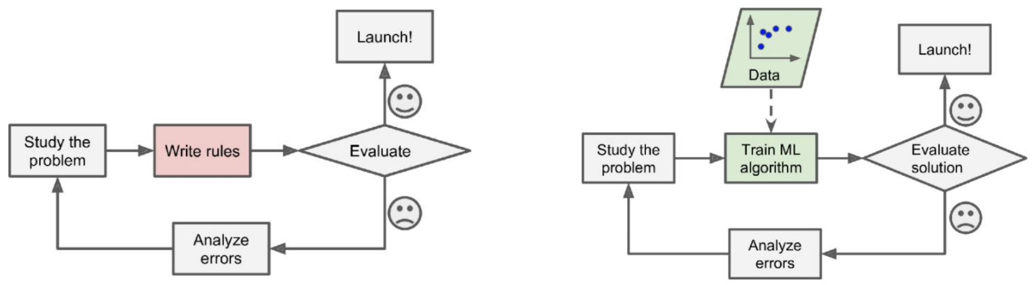
> [Machine Learning is the] field of study that gives computers the ability to learn without being explicitly programmed.[8]

What is the difference between Machine Learning and simple programming? The main difference is the approach of solving a given problem. Consider the general problem:

$$y = f(x) \tag{2.1}$$

where x is the input, y is the corresponding output and f a generic function. To solve the problem the computer need to learn, from the available data x, the best approximation of the function f. Classical approach consist in writing the best algorithm that can approximate as best as possible the true model represented with the fixed function f. The function f is considered fixed since the algorithm is composed by rules and steps that have the role to produce an output as close as possible to y. This result need to be achieved for every input and output couple (x, y). The rules and steps are fixed but, if the obtained results are too far from the expected one, they can be rewritten to obtain compatible results. The Machine Learning approach, as opposite to the classical one, is based on the available data and not on the algorithm. In this approach is not employed a fixed model (f

function) based on coding rules and steps, but it is applied a general model that can be tuned to achieve as best as possible his role (to obtain the desire results), by adjusting its parameters.
In figure 2.1 the two model's scheme are compared:



(a) Traditional **(b)** Machine learning

Figure 2.1: Comparison between traditional and machine learning approach.

A formal definition is provided by Tom Mitchel in 1997:

> A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E.[9]

In other words the strategy used in this context is to learn from experience E in order to improve performances related to a task T.

Let's now analyse why machine learning has becomes so used and powerful. Machine Learning allows to solve very complex problems, for which classical approach has no solution, providing a simplify solution achieving better performances. Another important feature is the *adaptivity*. Machine Learning programs can adapt to data changes or addition, and can manage large amount of data. Due to this features Machine Learning is used in several application fields i.e robotics, medicine passing, automation, biology, aerospace, finance etc. An example can be the detection of tumors in brain scans, self-driving cars, forecast company's revenue for the next year, virtual assistance and so on.

2.2 Evolution of the Machine Learning over the years

In 1943 Walter Pitts and Warren McCulloch wrote a paper about neurons. To show how they work they built a model using an electrical circuit and this work sanctioned the creation of the first Neural Networks.

Thanks to the discover of the Neural Networks, in the 1950 different applications are developed. In 1952 Arthur Samuel wrote a computer program, a game which played checkers, that can improve itself while it ran.
In 1958 was designed *Perceptron*, the first artificial Neural Network, by Frank Rosenblatt with the goal of recognize shape and pattern.
Another important breakthrough was, in the 1959, the creation of *ADELINE* and *MADELINE*, two Neural Network models built by Bernard Widrow and Marcian Hoff, with the role of detect binary patterns and eliminate echo on phone lines.

The progresses made by ANN's study were slowed down until 1970, when a new architecture, called Von Neuman architecture, was created. It was based on the stored-program computer concept, where instruction data and program data share the same memory space. This new architecture thanks to its simplicity and this new feature became the most popular methods to build programs.

The intense study of Neural Networks resumed in 1982, when John Hopefield creates a bidirectional line network and when, in 1986, was introduced the backpropagation algorithm that allows to train multiple-layered Neural Networks.

Another slow down occurs in '80s and '90s but at the end of 1990, precisely in 1997 and 1998 progresses were made.
The Deep Blue IBM computer had beaten the actual chess world record and at AT&T Bell Laboratories a research on hand-write digit recognition gave high accuracy results.

In 21st century the potential of Neural Network has been finally realized and this discipline took off. It allows to build very complex and deep networks that can solve difficult tasks.
In 2012 was developed GoogleBrain that was able to detect pattern in images and video thanks to a deep Neural Network. In the same year was created also AlexNet, which due to the use of GPU's and CNN won the ImageNet competition. The designers of AlexNet have created also the ReLU activation function that has contributed to the consistently improvement if the convolutional Neural Network's architecture.
In 2015 was developed ResNet one of the most used architecture for CNN.

2.3 Artificial Neural Network

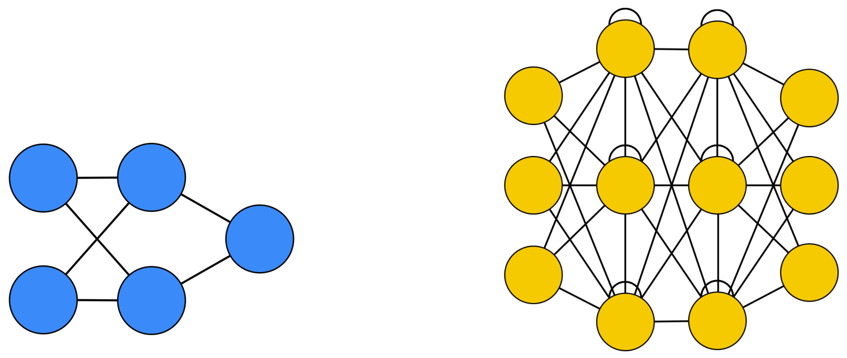
Artificial Neural Networks (ANN), also called Neural Networks (NN) is the most important branch of Deep Learning.
The invention of this discipline, as other many inventions, is inspired by the nature. In this case the starting point was the study of the biological neuron network of our brain.
This allusion to the brain's neurons implies only representative model since in the

years the ANN became very different from the biological one.

A Neural Network is composed by neurons, which constitute several layers belonging to three distinct types.
The *neurons* are the core units of the network and are organized in layers. Each neuron performs simply mathematical operations taking an input and producing an output that is feed to the other neurons.
The *input layer*, that is the first layer of the structure, acquires data from external sources.
The *hidden layers* perform various mathematical computation and cannot be seen from the outside of the network. The number of hidden layers can be one (Perceptron) or multiple.
The *output layer* is the last layer of the architecture and returns the obtained results.
The structure of the connections between layers can vary, usually it is a fully connected one where all the neurons of a layer are connected to all neurons of the following one.

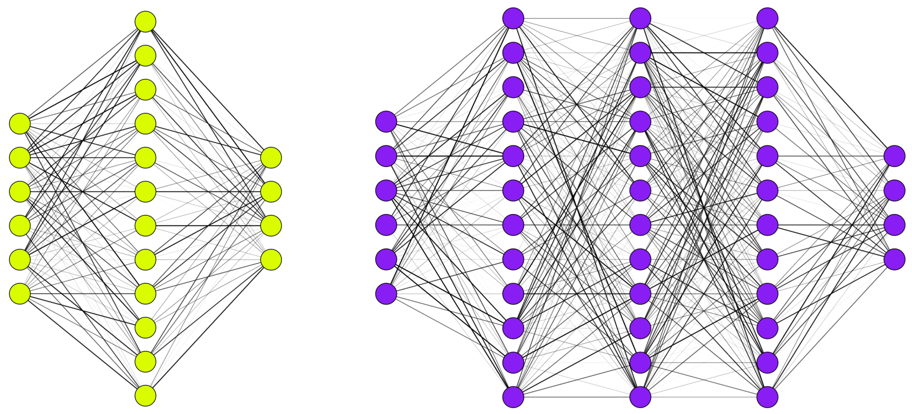
There can be identified two main types of Neural Networks: *FeedForward* and *FeedBack*.
The FeedForward Neural Networks are mostly employed in supervised learning, for example in image recognition and classification. The structure, as the name can suggest, does not present loops. The data and the computations are executed only in one direction, from input layer to output one. Instead the FeedBack Neural Networks are mainly used for storing purpose as in the Recurrent Neural Network (RNN) and they are composed by feedback loops.

Moreover, the NNs can be classified analyzing the number of hidden layers of which are composed. Two types of NNs are derived, *Shallow* and *Deep* Neural Networks. In Shallow NNs the number of hidden layers is at most two, instead in Deep NNs an higher number of hidden layers is supported, allowing the network to compute more complex tasks. In figure 2.2 is showed the difference between FFNN and FBNN and in figure 2.3 is visualized the difference between Shallow and Deep Neural Networks.
To understand how an ANN works we need at first to analyze its core unit, the neuron.



(a) Feed Forward (FF) **(b)** Feed Backward (FB)

Figure 2.2: Comparison between Feed Forward and Feed Back Neural Networks.



(a) Shallow Neural Network **(b)** Deep Neural Network

Figure 2.3: Comparison between Shallow and Deep Neural Networks.

2.3.1 Neuron Model

Before entering in detail analyzing the artificial neurons should be provided a quick overview of the biological neuron. The *neuron* is a cell present in humans and animals brain and is composed by:

- **Soma**: is the cell body. It contains the nucleus and the most important component of the cell;
- **Dendrites**: are extensions of the soma and can have several branches;

- **Axon**: is the longest extension of the soma;

- **Telodendria**: are the branches at the extremity of the axon;

- **Synapse**: structure at the tip of each telodendria. It acts as interconnection between the dendrites or soma of other neurons.

The neuron can produce short electrical signals, called *action potentials*, which travel through the axon to the synapses that release the neurotransmitters. If the neuron receive a large enough quantity of neurotransmitters, in small amount of time, it produces its own electrical impulse that is transmitted along its axon to other neurons. In figure 2.4 is represented the schematic of a biological neuron.

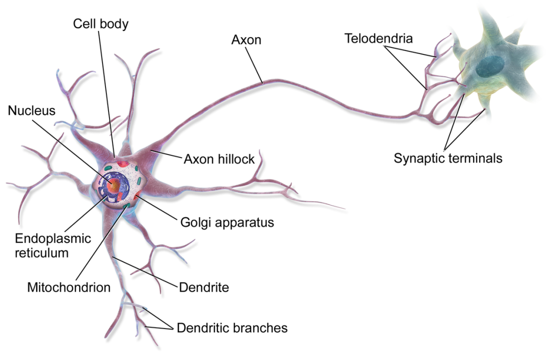


Figure 2.4: Biological neuron. [8]

Artificial neuron – McMulloch and Pitts

The fist model of an artificial neuron was proposed by McMulloch and Pitts in 1943.
The artificial neuron structure is very simple, it has one or more binary inputs and one output that is activated only when the number of active inputs are above a certain threshold. Despite the simplicity of this model, is possible to build any possible logical proposition. To prove that, in figure 2.5 are showed some ANNs that performs various binary operations. This operations are possible under the assumption that a neuron is active only when the input threshold is equal to two, this mean that at least two inputs need to be active to have an output result.

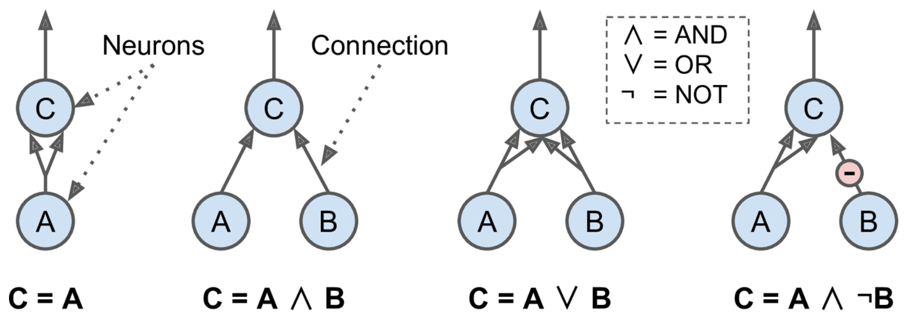


Figure 2.5: Artificial Neural Networks performing binary operations.[8]

The Perceptron

The Perceptron is one of the basic and simple ANN architectures. It is based on the *Linear Threshold Unit* (LTU) model (figure 2.6).

The LTU is a generalization of the previously explained neuron model. As input, it has one or more values $(x_1, x_2, \cdots, x_N)$ which can assume real values and only binary one, and one output. Each input connection has a related weight $(w_1, w_2, \cdots, w_N)$. The LTU neuron perform a weighted sum of the inputs and to them applies an activation function φ.
The output can be written as:

$$y = \varphi\left(\sum_{i=1}^{N} w_i * x_i\right) = \varphi(w * x) \tag{2.2}$$

where x is a column vector containing the input values and w is a row vector containing the connection weights. In Perceptron the most used activation function is the *Heaviside step* but another function is employed, the *Sign*.

$$H(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases} \tag{2.3}$$

$$sng(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases} \tag{2.4}$$

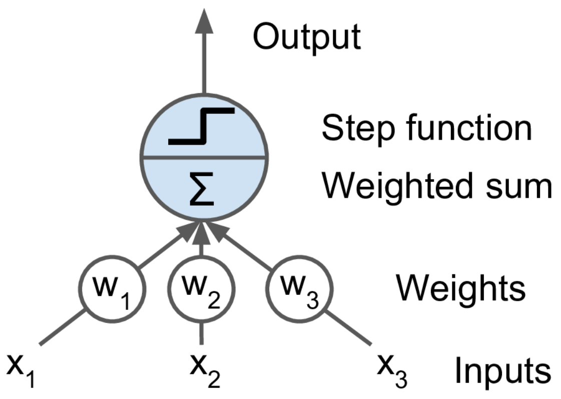


Figure 2.6: Linear Threshold Unit model.[8]

As mentioned above the Perceptron is based on a single layer of LTU that acts as output layer. The input layer is composed by the input neurons and the bias term ($x_0 = 1$) contained in a bias neuron. The network is fully-connected, this means that each input is connected to each neuron.
The Perceptron model can be represented by the function:

$$y = \varphi\left(\sum_{i=1}^{N} w_i * x_i + w_0\right) = \varphi(W * X + b) \tag{2.5}$$

where X is the input feature matrix, W the weight matrix, b the bias vector, which contains all the weight between the bias and the artificial neurons and φ the activation function.
In figure 2.7 we can observe the model of a Perceptron having two inputs and three outputs.

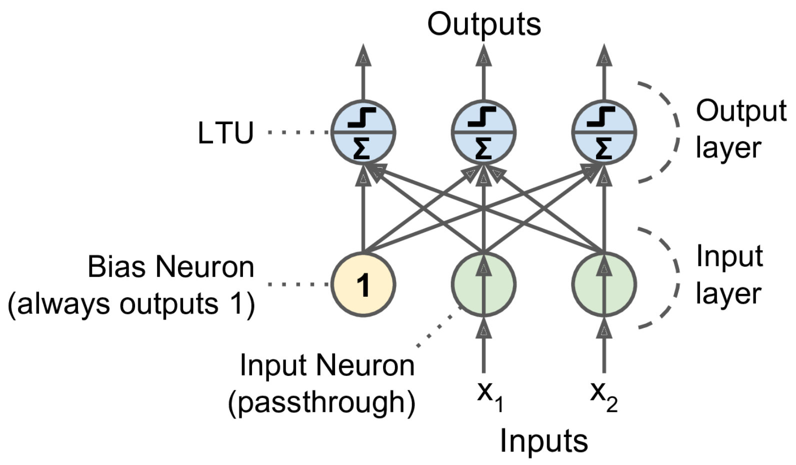


Figure 2.7: Perceptron model.[8]

This kind of Perceptron model, having only one layer of neurons, has some limitations, as highlighted in 1969 by Minsky and Papert on the monograph Perceptron. It can't solve some simple problems, as the XOR classification problem and can only classify linear separable features.
Researchers did not give up and found a way to overcome these limitations by stacking multiple Perceptrons in a new ANN model called Multi Layer Perceptron (MLP).

Multi Layer Perceptron - MLP

The structure of the MLP is the following:

- One *input* layer;
- One or more *hidden* layers;
- One *output* layer.

The hidden layers are formed by LTUs.
We can denominate the layers on the basis of the proximity to the input or output layer: *lower* layers are close to the input, *upper* layers are close to the output. The input and all the hidden layers have a bias neuron. The structure of the network is fully connected. An analysis, based on the number of hidden layers, can determine the type of the network. If the ANN contains a big stack of hidden layers the network is called *Deep* Neural Network (DNN), instead if it has a small number of hidden layers is called *Shallow* Neural Network. This kind of ANN allows to perform complex task and solve trivial problems. In figure 2.8 can be seen a MLP composed of two inputs neurons, one hidden layers with four neurons and three output neurons.

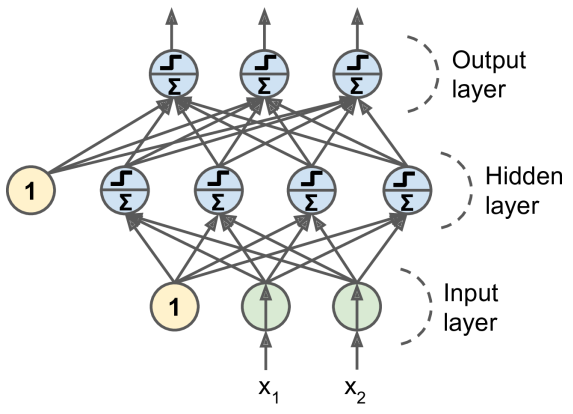


Figure 2.8: Multi Layer Perceptron model.[8]

Despite this NN structure has been found, researchers were not able to train it successfully until in 1986, Rumelhart, Hinton and Williams published a paper on the *backpropagation* training algorithm. This algorithm allows to compute automatically the gradients in two passes through the network, one forward and one backward. The *forward pass* makes a prediction and measures the error. The *reverse pass* goes through each layer to measure the error contribution of each connection. The last step, the *Gradient Descent* step, tries to reduce the error finding how to adjust each connection weight and bias term. Those steps are repeated until the network reaches an optimal solution.
The introduction of the backpropagation algorithm allows to move from a first generation of neurons to a new one.

Sigmoid Neuron The introduction of the backpropagation algorithm makes progresses in the study of the Artificial Neural Networks giving substantially benefits. To obtain the best performances out of the algorithm some changes on the MLP architecture have been done. In the first MLP model the activation function φ was a binary function. One of the most used was the Heaviside step which outputs either 0 or 1. To correctly implement the backpropagation algorithm was introduced a new continuous activation function, the Logistic function, also called *Sigmoid*:

$$\sigma(z) = \frac{1}{1 + e^{(-z)}} = \frac{1}{1 + e^{(-W*X+b)}} \tag{2.6}$$

The Sigmoid function permits the building of any continuous function using a single hidden layer. The artificial neurons, as consequence, can handle analog values. Figure 2.9 shows the Sigmoid function of z.

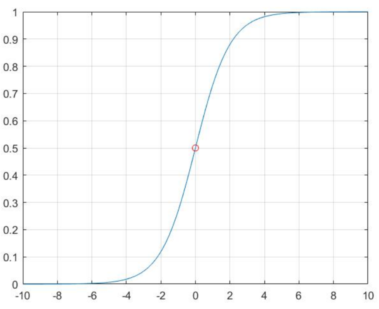


Figure 2.9: Sigmoid function.

As can be seen the Sigmoid function is a sort of smooth version of the Heaviside step. It is differentiable at any point of its domain and the Gradient Descent can make progress at each step. The computation of the gradients and the learning process is more accurate and gives better results. The Sigmoid function has a disadvantage since for very high or low values of z the vanishing gradient problem may show up. It involves in the inability of further learning or reach an accurate prediction in a reasonable time. Due to its characteristics Sigmoid function is one of the most used activation functions in Neural Networks.

Activation functions

Tanh – Hyperbolic tangent It is a S-shaped function, similar to the Sigmoid one. The main difference is that it ranges from -1 to 1 and not from 0 to 1. The Tanh function is continuous and differentiable at any point of its domain. The outputs of each layer are zero centered, this leads to a fast convergence. The hyperbolic tangent is showed in figure 2.10, while its equation is:

$$tanh(z) = 2 * \sigma(2z) - 1 = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.7}$$

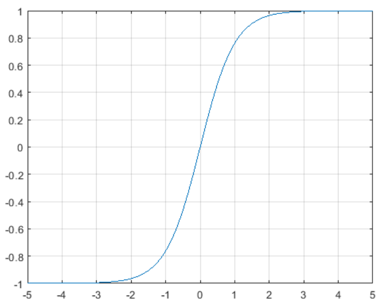


Figure 2.10: Hyperbolic tangent function.

ReLU – Rectified Linear Unit The ReLU is a continuous function, differentiable everywhere except in z = 0. It is represented by the equation:

$$ReLU(z) = max(0, z) \tag{2.8}$$

The output of this functions can be: z if $z > 0$, with derivative equal to 1; 0 if $z < 0$, with derivative equal to 0. Despite it has a not differentiable point, most of the units have $z > 0$. This allows to have almost everywhere a positive slope that speeds up the computations.
It has a disadvantage called "Dying ReLU problem". When inputs approach zero,

or are negative values, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn anymore.

ReLU function become the default choice in machine learning and deep learning problems. In figure 2.11 is visualized the ReLU function.

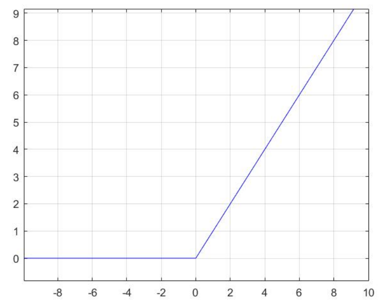


Figure 2.11: Rectified Liner Unit function.

Leaky ReLU The Leaky ReLU is a variation of the ReLU function and solves the dying ReLU problem. It has a small positive slope when z is less or equal to 0. This slope makes possible backpropagation also for negative values. The slope term is represented by α and usually has a low value as 0.01.
The Leaky ReLU performs better than the original function but, since it has unreliable prediction for negative values, is not often used. In figure 2.12 is showed the Leaky ReLU and its equation is:

$$LeakyReLU(z) = \begin{cases} z & z < 0 \\ \alpha z & z \geq 0 \end{cases} \tag{2.9}$$

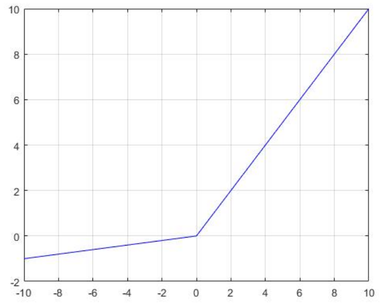


Figure 2.12: Leaky ReLU function.

Softmax The Softmax activation function is usually employed in the output layer of a classification problem. It is a generalization of logistic regression and it is used for multi-class classification. It get the input values, it transform them into values ranging from 0 to 1 and gives as result a probability. This probability represents the percentage that an input belongs to a specified class. The sum of all the probabilities must be 1. The equation is similar to the Sigmoid function:

$$Softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.10}$$

The term z_i refers to the elements of the input vector z. The denominator performs as normalization term which guarantees that the sum of all outputs is equal to 1. The term K refers to the number of classes.

2.3.2 Architecture of a Neural Network

In the previous chapter has been analyzed the structure of a neuron but is good to understand also the architecture of the entire Neural Network. The artificial neuron performs a weighted sum of all the inputs, adds a bias term, and applies to the result an activation function σ:

$$z = w * x + b \tag{2.11}$$

$$y = \sigma(z) = a \tag{2.12}$$

X refers to a column vector containing the inputs values and w to a row vector containing all the weights. The neurons are organized in layers and to distinguish the elements of the equation is introduced the following notation:

$$a_i^{[l]} \tag{2.13}$$

The apex l represent the layer which the neuron belongs to, the subscript i refers to the corresponding neuron's parameter. The letter L is used to denominate the total number of layers.

Once the notation is introduced can be wrote a general equation denoting the behavior of a generic layer l of a NN:

$$z^{[l]} = w^{[l]} * a^{[l-1]} + b^{[l]} \tag{2.14}$$

$$a^{[l]} = y^{[l]}(z^{[l]}) = \sigma(z^{[l]}) \tag{2.15}$$

- $a^{[l-1]}$: vector of the activation function referring to the previous layer. It is the input of the lth layer. The first input layer is denoted by $x = a^{[0]}$;
- $w^{[l]}$: matrix containing the weights of the current layer l;
- $b^{[l]}$: vector containing the biases referring to the current layer l;

- $a^{[l]}$: vector of activation function referring to layer l.

Analyzing the NN reported in figure 2.13, on the basis of what is said above, we can get that: there are four layers (L = 4), the three hidden layers have five neurons each and the input layer which can be represented as x, is not counted in the total numbers of layers. Below are reported the corresponding layers equations. For convenience, since the mathematical statements are similar, are reported only the first and the output layer equations:

$$x = a^{[0]} \tag{2.16}$$

$$z^{[1]} = w^{[1]} * x + b^{[1]} \tag{2.17}$$

$$\dots \tag{2.18}$$

$$z^{[4]} = w^{[4]} * a^{[3]} + b^{[4]} \tag{2.19}$$

$$a^{[4]} = \sigma(z^{[4]}) \tag{2.20}$$

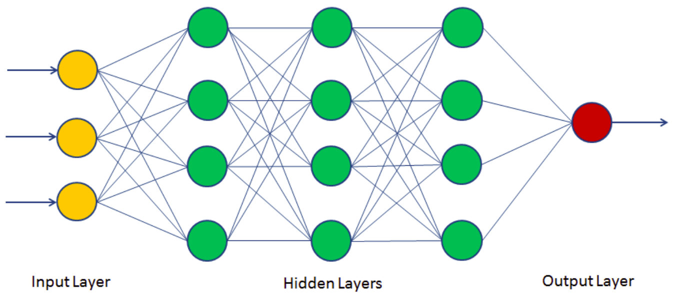


Figure 2.13: Simple Neural Network.

2.3.3 Training a Neural Network

In this section it is analyzed the most important step towards the development of a Neural Network: the training phase. It is important to understand how to choose the right training algorithm, the hyperparameters and other features to develop in a quick way the most appropriate model.

The first and most essential element that must be used to train a network is the dataset. The dataset contains n known values represented as input-output couples (x, y) which are needed to tune the biases and weights parameters.

$$\{(x_1, y_1), (x_2, y_2), \cdots, (x_{(n-1)}, y_{(n-1)}), (x_n, y_n)\} \tag{2.21}$$

The training process is developed in two phases. First of all, the input X is propagated through the network until a resulting $\hat{y}$ is obtained. Then, the error between the desired output y and the predicted one $\hat{y}$ is calculated and minimized. Finally, biases and weights are updated. To train a network, in summary, we need to optimize a function called cost or loss function.

Cost function

The cost function is a measure of how good is going the learning phase, since it is calculated and updated constantly during the training of the network. To be computed, it needs the predicted output $\hat{y}$ and the training examples. They are contained in the train dataset, which is a portion of m elements taken from the entire dataset. The input x_i passes though the network and gives as result the predicted output $\hat{y_i}$. Then, the cost function $C(w, b)$ is computed by calculating the error between the actual output $\hat{y}$ and the desired one y. This error quantity must be minimized and this reflects in the calculation of the final biases and weights values.
A typical loss function is the Mean Square Error (MSE) also known as L2 loss:

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y_i})^2 \tag{2.22}$$

Minimizing the MSE means trying to obtain an MSE approximately equal to 0. To achieve this result $\hat{y_i}$ must be as close as possible to $\hat{y}$.

Other cost functions

Mean Absolute Error The MAE on the contrary to MSE does not take the square of the difference between y and $\hat{y}$ but only the difference. Due to this aspect it is also called L1 loss.

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|(y_i - \hat{y_i})| \tag{2.23}$$

It is the mean of the absolute differences among predictions and expected results, where all individual deviations have even importance.

Cross Entropy – CE The Cross Entropy takes an average between the classes errors, which measure the distance between the true value and the predicted one. The Cross Entropy can be defined as the difference between the probability distribution of the true and the predicted output, remembering that they range from 0 to 1. It is the standard choice for classification problems since is very effective due to the strong use of the probability.

The equation representing the Cross Entropy is:

$$CE = \sum_{i=1}^{C} y_i log(\hat{y}_i) \tag{2.24}$$

C refers to the number of classes, y_i to the true output and $\hat{y}_i$ to the predicted output.

Categorical Cross Entropy - CCE The Categorical Cross Entropy, also called Softmax Loss, is a Softmax activation plus a Cross-Entropy loss function. It is used for multi-classification problems where only one result can be correct, in other terms the predicted output can belongs only to one class.
The Categorical Cross Entropy can be described by those equations:

$$f(z)_j = \frac{e^{z_j}}{\sum_{j=1}^{C} e^{z_j}} \tag{2.25}$$

$$CE = \sum_{i=1}^{C} y_i log(f(z)_i) \tag{2.26}$$

Binary Cross Entropy - BCE The Binary Cross Entropy, or Sigmoid Cross Entropy, is another particular case of Cross Entropy loss function. As can be suggested by the name is a Sigmoid activation plus a Cross Entropy loss function. Contrary to CCE it is used in multi-classification problems where the predicted output can belong to different classes, since each output value is independent from the others in terms of probability.
To resolve the problem it must subdivided into C binary problems where $\hat{y}_i$ is related to the probability of belonging to class i, and 1-$\hat{y}_i$ to the probability of not belonging to that class. Then, is applied the Cross Entropy to $\hat{y}_i$ and 1-$\hat{y}_i$ obtaining the following equation:

$$CE = -\sum_{i=1}^{2} y_{ij} log(\hat{y_{ij}}) = -y_i log(\hat{y}_i) - (1 - y_i) log(1 - \hat{y}_i) \tag{2.27}$$

Since y_i can assume the value 0 or 1 the loss can be rewritten as:

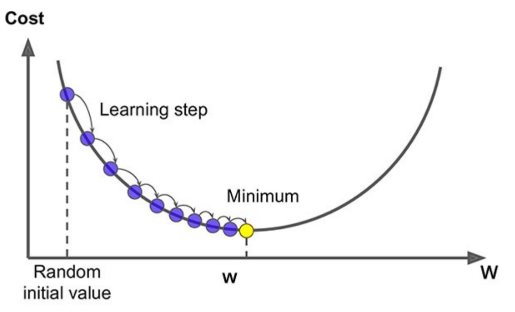
$$CE_i = f(x) \begin{cases} -log(\hat{y}_i) & y_i = 1 \\ -log(1 - \hat{y}_i) & y_i = 0 \end{cases} \tag{2.28}$$

To compute the BCE it is computed the sum of the different CE_i contributions:

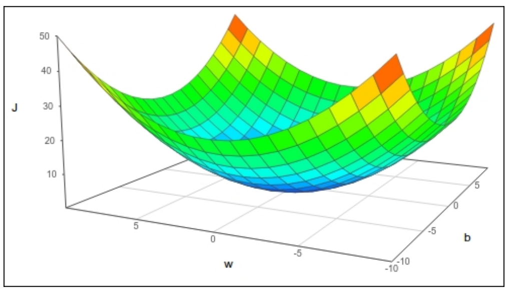
$$CE = \sum_{i=1}^{C} CE_i \tag{2.29}$$

Gradient descent

All the cost functions treated above are supposed to be minimized, but the minimization is not an easy operation since the number of parameters, that is equivalent to the number of dimensions, are usually higher that two.
In figure 2.14 can be seen the minimization of a cost function with one and two parameters. It is easily understandable why having more than 2 parameters could be a problem.



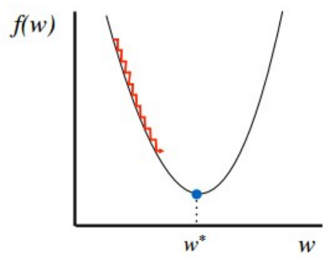
(a) One parameter minimization problem [8]



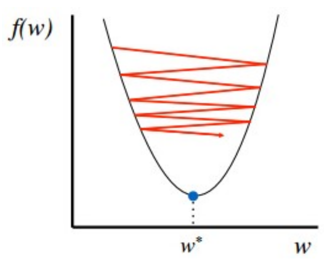
(b) Two parameters minimization problem

Figure 2.14: Comparison between Gradient Descent minimization with a different number of parameters.

To solve this problem are used some specific algorithms called *optimizers*. Those algorithms allow to update the parameters toward the minimum direction. The simplest optimizer is the Gradient Descent that is capable of tuning the parameters, in a iterative fashion, to minimize the cost function. The parameters that are going to be minimized are contained in the vector θ. The gradient descent algorithm works in a simple way: at first the vector θ is filled with random values, then, it measures the gradient of the cost function with respect to the vector θ. The algorithm updates the values of θ and measures the gradient based on the direction of the descending gradient. It stops only when the gradient is 0, this mean that the minimum is found. The final value is founded gradually, minimizing step by step the cost function until there is a convergence to zero. The size of the steps is defined by the *learning rate* hyperparameter. If the learning rate is too small the time of convergence can be very high (2.15a) instead if it is too high the algorithm pass through the minimum jumping to the other side of the curve and may cause divergence of the algorithm (2.15b). A trade-off must be found.



(a) Small learning rate



(b) High learning rate

Figure 2.15: Learning rates comparison.

Other minimization problems can arise on the basis of the function convexity. Founding the minimum of a convex function can be difficult since the Gradient Descent algorithm can trap in a local minima or remain for a long time on a plateau region. Figure 2.16 provides a visual representation of these problems.

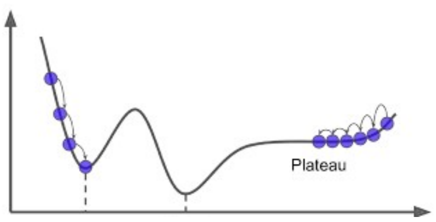


Figure 2.16: Convex function problem

Below is analyzed in detail the Gradient Descent algorithm.
At first the cost function $C(\theta)$ is considered. It depends on m parameters $\theta = (\theta_1, \theta_2, \cdots, \theta_m)$. The cost function variation has to be calculated applying a small change of the quantity θ. This is called partial derivative and is used to obtain the gradient of the function.

$$\Delta C \approx \frac{\partial C}{\partial p_1} * \Delta\theta_1 + \frac{\partial C}{\partial p_2} * \Delta\theta_2 + ... + \frac{\partial C}{\partial p_m} * \Delta\theta_m = \nabla C * \Delta\theta \tag{2.30}$$

The term $\nabla C \equiv (\frac{\partial C}{\partial p_1}, \frac{\partial C}{\partial p_2}, \cdots, \frac{\partial C}{\partial p_m})$ refers to the gradient of the cost function with respect to the parameters computed in the actual point.
The parameters are tuned to decrease the cost function following the decreasing gradient direction, then the term $\Delta\theta$ is chosen in such a way the variation of the gradient ∇C is negative. This is possible by choosing $\Delta\theta$ as:

$$\Delta\theta = -\eta\nabla C \geq \nabla C * \Delta\theta = -\eta\|\eta\nabla C\|^2 < 0, \eta > 0 \tag{2.31}$$

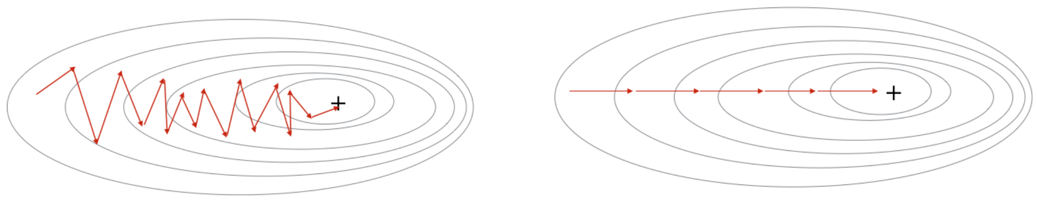
The quantity η is called learning rate and is always a positive value. The choice of the learning rate value is tricky since an high value may lead to divergence and a too small value may lead to long training time, that implies a low convergence. The learning rate can be changed during the training process to achieve an optimal solution.

Other optimization algorithms

The Gradient descent is one of the optimizers that can be used to train a Neural Network. To efficiently train a ANN, the optimizer need to be chosen wisely to obtain the best solution in the least amount of time. Some of the most used optimizer are reported above.

Batch Gradient Descent The Batch Gradient Descent is very similar to the Gradient Descent algorithm, the main difference is the size of the evaluation step. To compute the gradients, at each step, it is considered the whole training set instead of a part of it. This approach leads to high training time specially for large datasets.

Stochastic The Stochastic algorithm is the opposite with respect to the Batch Gradient Descent since the size of each step is one. The Stochastic Gradient Descent takes one random instance among the training set and evaluate the gradient. The advantage to use little amount of data is the speed of the algorithm that can be very fast. It also avoids memory problem since only one instance is present in the memory at each iteration. Using this algorithm huge datasets can be trained without problems.
The main drawback of this algorithm is the non-regularity. The cost function not decreases until the minimum but bounces up and down decreasing only on the average. The final result is good but not optimal because when the algorithm is close to the minimum it continue to bounce around it. This irregularity helps in founding a global minimum but not the optimal one. The figure 2.17 compares the behavior of the GD and the SGD.



(a) Stochastic Gradient Descent **(b)** Gradient Descent

Figure 2.17: Stochastic and Gradient Descent algorithm comparison.

Mini-batch Gradient Descent Mini-Batch Gradient Descent is the algorithm that lays between the GD and the Stochastic GD. The Mini-Batch GD at each step computes the gradients using a set of given instances taken randomly from the training set. The set of instances is called *mini-batch* and its dimension varies from two to the dimension of the training set minus one. Usually the batch size is ranging from 32 to 256. The main advantage with respect to Stochastic GD is the increase of performance when a GPU is utilized. Another important aspect is reduction of the distance between the optimal and the obtained minimum. The figure 2.18 compares the behavior of the Stochastic GD and the Mini-Batch GD.

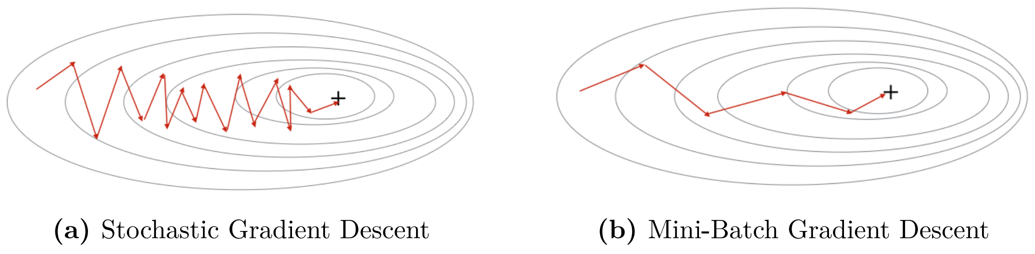

(a) Stochastic Gradient Descent **(b)** Mini-Batch Gradient Descent

Figure 2.18: Stochastic and Mini-Batch Gradient Descent algorithm comparison.

Momentum The Momentum is a method that reduces the oscillation amplitude of GD taking into accounts the past gradient values to smooth out the update. The direction of the previous gradient is stored and used to calculate the current gradient. The updating of the gradient can be described by the equations:

$$m_t = \beta m_{t-1} + \beta \nabla C \tag{2.32}$$

$$\Delta\theta = -m_t \tag{2.33}$$

The term βm_{t-1} refers to the fraction β of the previous step. β is called *momentum term* and has a positive value lower than one, usually equal to 0.9.

Until now the optimizer parameters are considered constant and can be changed only at the beginning of the training process. There are some optimizers called *adaptive optimizers* which parameters constantly adapt for each and every weight and bias computation. Above are analyzed two of them: the RMSprop and the Adam algorithm.

RMSprop In the RMSprop algorithm the learning rate is changed during each parameter update. The update takes into account the previous values of the parameters. Each weight has its own cache value which, for simplicity, is called E_t.

$$E_t = \gamma E_{t-1} + (1-\gamma)(\nabla C)^2 \tag{2.34}$$

The term E_t refers to the new cache value, instead E_{t-1} to the previous one. The parameter γ is called *decay rate* and is always lower than one, usually 0.9 or 0.99.

The working principle is the following: if a weight is updated a lot also the cache value increases by a big quantity. As consequence the learning rate is decreased and with it also the update magnitude of the weight. On the contrary, if the weight does not undergo any considerable update, its cache value decreases and the learning rate increases forcing bigger updates. The learning rate is changing constantly with respect to the weight updating but does not decay quickly enabling a long train process.
The last step is the updating of the parameters vector.

$$\Delta\theta = -\frac{\gamma}{\sqrt{E_t + \epsilon}}\nabla C \tag{2.35}$$

At denominator, under the square root, the term ϵ avoids a division by 0 ensuring a valid result.

Adam Adam algorithm, also called *Adaptive Momentum Estimation* algorithm, is an adaptive optimizer. It is a combination of the RMSprop and Momentum methods. First of all it calculates an exponentially weighted average of past gradients, which represents the momentum.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla C \tag{2.36}$$

$$\hat{m}_t = \frac{m_t}{(1-{\beta_1}^t)} \tag{2.37}$$

Then, it is calculated an exponentially weighted average of the squares of the past gradients, which represents the accumulated cache.

$$E_t = \beta_2 E_{t-1} + (1-\beta_2)(\nabla C)^2 \tag{2.38}$$

$$\hat{E}_t = \frac{E_t}{(1-{\beta_2}^t)} \tag{2.39}$$

Finally, the parameters are updated using the information obtained in the previous steps.

$$\Delta\theta = -\frac{\gamma}{\sqrt{\hat{E}_t + \epsilon}}\hat{m}_t \tag{2.40}$$

The terms $\hat{E}_t$ and $\hat{m}_t$ refer to the corrected cache and momentum. Those terms are introduced to reduce the effect of the biases β_1 and β_2 in the first computation steps because they are close to 1. The suggested values for the parameter β_1, β_2 and ϵ are 0.9, 0.99 and 1e-8 respectively.
Adam performs better that other optimizer and is choose as default when a Neural Network need to be trained.

Backpropagation

When Gradient Descent or other optimizers are applied to a NN they must compute the partial derivative of the cost function with respect to the m parameters.
A NN is composed by several layers which contain a variable numbers of neurons. Each neuron introduces a weight and a bias. This leads to an huge numbers of variables and an high computational cost.

The Backpropagation algorithm solves this problem reducing the computation time.
The goal of this algorithm is to calculate the partial derivatives $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ of the cost function with respect to any weights and biases. It calculates the derivative, layer per layer, and reuses the computations performed in the last layers to derive the derivatives of the previous layers. Those operations are performed in a iterative fashion.

Before analyzing the fundamental equations of the Backpropagation algorithm, to better understand them, the Hadamard product has to be introduced.
The Hadamard product, or Shur product, is the element wise product of two vectors.
For example multiplying a vector a and vector b element wise the obtained result is the following:

$$a \bigodot b = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \bigodot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1*3 \\ 2*4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix} \tag{2.41}$$

Backpropagation fundamental equations The four fundamental equations that constitute the Backpropagation algorithm are reported below.

$$\delta^L = \nabla_a C \bigodot \sigma'(z^L) \tag{2.42}$$
$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \bigodot \sigma'(z^l) \tag{2.43}$$
$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.44}$$
$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.45}$$

First equation The first equation computes the output layer error δ_j^L.

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \tag{2.46}$$

where:

- $\frac{\partial C}{\partial a_j^L}$ measures how fast the cost is changing as function of the j^{th} output activation. Less is the dependence of C on the output j and smaller δ_j^L will be;
- $\sigma'(z_j^L)$ measures how fast the activation function σ is changing at z_j^L.

Those terms can be easily computed and the form $\frac{\partial C}{\partial a_j^L}$ is strictly related to the used cost function.
The above equation can be rewritten in a matrix form.

$$\delta_j^L = \nabla_a C \bigodot \sigma'(z_j^L) \tag{2.47}$$

Where $\nabla_a C$ is a vector containing all the partial derivatives $\frac{\partial C}{\partial a_j^L}$ and $\odot$ is the Hadamard product.

Second equation The second equation computes the output error δ^l in terms of the next layer error δ^{l+1}. Using this method is possible to go forward in the network computing layer by layer the output error. As example, starting from δ^l it is possible to calculate the errors δ^{l-1}, δ^{l-2}, and so on.

Third equation The third equation describes the changing rate of the cost with respect to any bias in the network.

$$\frac{\partial C}{\partial b_j^L} = \delta_j^L \tag{2.48}$$

Fourth equation The fourth equation computes the partial derivatives of the cost function with respect to the weights $\frac{\partial C}{\partial w_{jk}^l}$. The calculation of the partial derivatives is done using the the terms δ^l and a^{l-1}.
This equation introduces an aspect that need to be taken into account in the calculation of the Gradient Descent. If the activation of the input neuron is small also the gradient will be small reflecting into a small change of the weight during the computation of the Gradient Descent algorithm. In general, output weights, from low activation neurons, learn slowly.

2.4 Convolutional Neural Network

Convolutional Neural Networks (CNNs) have been used since the 1980s and their model is derived from the study of the visual cortex. CNNs are specially used for image recognition purpose but can handle many other tasks, as voice recognition and natural language processing. In last years this field grows exponentially thanks

to the increasing of the amount of training data and of the computational power. In 1958 and 1959 the scientists Hubel and Wiesel found that some neurons in the visual cortex had a local receptive field, which were able to react to simple and complex patterns. Some neurons can react to horizontal lines, others to vertical one and the neurons with large receptive field can react to more complex patterns that are a combination of the lower level one. The authors understood that there were an architecture in which higher level neurons are feed by the outputs of the lower level ones.
The study of the CNNs continues over the year until in the 1998 Yann LeCun did an important step creating the LeNet-5, which has introduced some important building blocks: the *convolutional* and the *polling* layers.

2.4.1 Convolutional layers

In this chapter is analyzed the principal building block of a CNN, the *convolutional* layer. CNN uses partially connected layers, this means that the neurons are not connected with all the pixels of the image, like in a fully connected architecture, but only with pixels that are in the receptive field. This is reducing the number of connections that are further stretched in the next convolutional layer since only the information coming from some part of the input matrix are elaborated. This architecture allows to include low-level features into larger high-level ones. For example the first layer can recognize only vertical and horizontal lines, that in the higher level are part of some geometric forms, like a rectangle, and so on until a more complex patter is recognized like a person.

2D convolution

The layers in a CNN are two dimensional since we are referring to images. The inputs are organized in 2D matrix with dimension $W \times H$ pixels, where W is width and H height. Each neuron of the hidden layer is related only to a portion of the input neuron area called *receptive field.* The receptive field has shape $f_w \times f_h$ and is usually a square matrix, then the shape can be rewritten as $f \times f$.

The operation implemented by each neuron is a weighted sum of the inputs. The result is fed to an activation function. The size of the receptive field affects the number of input features that are combined to obtain the output. This happens because each neuron has the same number of weights of the field.
The weights are organized in a matrix called *filter kernel* which is applied to the input. The obtained result is a 2D matrix, an image. The dimension of the output 2D matrix can be calculated using the following equation.

$$W \times H * f \times f \rightarrow (W - f + 1) \times (H - f + 1) \tag{2.49}$$

Where:

- W is the width of the input image;
- H is the height of the input image;
- f is the dimension of the filter;

In other words to the input image is applied a filter that slides from left to right, top to bottom applying the convolution operation calculating the output result. The following image is a visual representation of the convolution process, the input is a 5×5 matrix, the filter has dimension 3×3 and the obtained output is a 3×3 matrix.

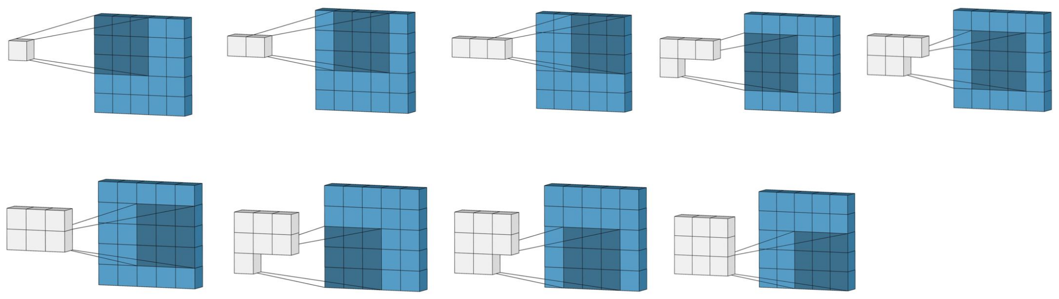


Figure 2.19: Convolution of a matrix 5×5 with a filter 3×3.

The dimension of the output matrix can be modified using two techniques that are commonly used in convolutional layers: the *padding* and the *striding*.

Padding

During the convolution process the edges are smoothed since the output size is reduced. Usually the output size must be equal to the input one. To solve this problem the edges are *pad* with some extra pixels, with value usually equal to 0. This technique is called *zero padding*.
The edge now is centered with respect to the kernel and this produced an output image with the same size of the input one. This technique can be summarized using the equation:

$$W \times H * f \times f \rightarrow (W + 2p - f + 1) \times (H + 2p - f + 1) \tag{2.50}$$

The term p refers to the padding.
To have *same padding*, that means having the input dimension equal to the output one, the value of p must be:

$$p = \frac{f-1}{2} \tag{2.51}$$

Considering the previous example (2.19) adding a pad of $p = \frac{3-1}{2} = 1$, the output dimension is:

$$5 \times 5 * 3 \times 3 \rightarrow (5 + 2 - 3 + 1) \times (5 + 2 - 3 + 1) = 5 \times 5 \tag{2.52}$$

As can be noticed, the output matrix has the same dimension of the input one.

Striding

In Convolutional Neural Networks, when the number of channels is increased, the size of the matrices is usually reduced. This implies that the size of the output need to be smaller wit respect to the input one.
The stride technique is based on shifting the kernel by more than one pixel at a time. This allows to downsize the output by a factor equal to the size of the stride.

$$W \times H * f \times f \rightarrow \left\lfloor \frac{W + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{H + 2p - f}{s} + 1 \right\rfloor \tag{2.53}$$

The term s refers to the stride parameter.

The input layer is connected to the hidden layer through a map called *feature map*. A convolutional layer can detect several features having multiple feature maps stacked together. The result are several matrices which underlines these features.

Below are analyzed some practical examples that can be applied to image recognition: the detection of a vertical or a horizontal edge.
The input dimension is equal to 6×6, the kernel size is equal to 3×3 and the output size is equal to 4×4 with no padding and stride $s = 1$. The matrices that refer to the vertical and the horizontal line detector are very similar as can be seen in figure 2.20.

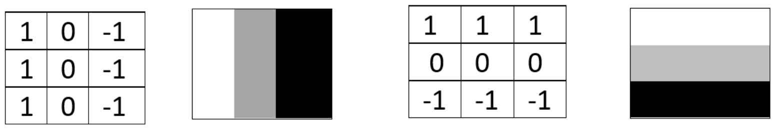


Figure 2.20: Vertical and horizontal edge detector matrices.

The vertical edge detection allows to highlight the areas in the image that presents vertical lines and to obtain a new image where this feature are emphasized. The same happens when a horizontal line detector is applied, obtaining an image that emphasizes the horizontal lines. Applying a vertical and an horizontal line detector to the same image leads to the production of two different images. Figure 2.21

shows the difference between the application of a vertical and an horizontal edge detector to the same image.

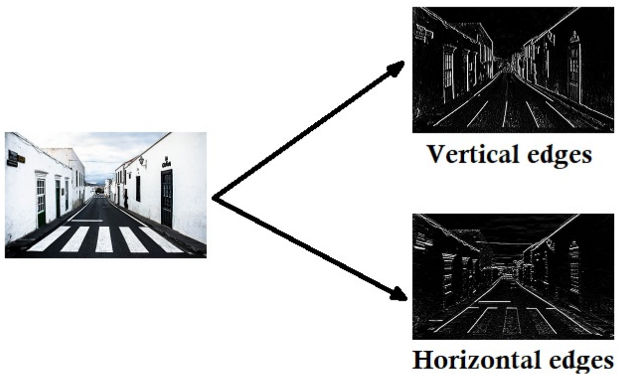


Figure 2.21: Vertical and horizontal edge detector applied to the same image.

3D convolution

An RGB image represents a 3D input matrix since there is one channel per color. The number of channels is represented by the last dimension number: $(W \times H \times C)$, where W stands for the Width, H for the Height and C for the number of channels. To have a correct convolution between the input image and the kernel, the corresponding channel number must be the same. Each filter of the kernel convolute with the input image channels. The pre-processed channels are then summed together and an activation function is applied. To better understand this process a convolution between a RGB image of dimension 6 and a filter of dimension $3 \times 3 \times 3$ is visualized in figure 2.22.

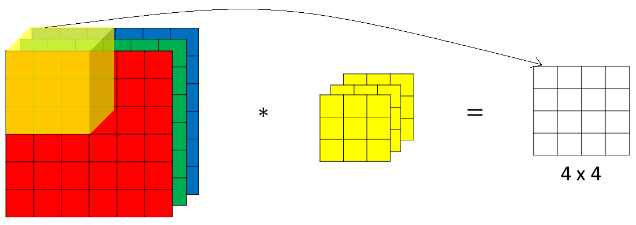


Figure 2.22: Convolution between an RGB image and a filter $3 \times 3 \times 3$.

The obtained result is a 2D image, and not a RGB one, with dimension 4×4. The output shape can be calculated using the formula:

$$W \times H \times C * f \times f \times C \rightarrow (W - f + 1) \times (H - f + 1) \tag{2.54}$$

Until now, each convolutional layer is considered having only one filter, but in general, a convolutional layer has multiple filters. The number of filters is proportional to the number of feature maps since each filter has one feature map. A convolutional layer, applying multiple filters, can detect at the same time several features of the input image. The above equation can be rewritten to obtain:

$$W \times H \times C * f \times f \times C \rightarrow (W - f + 1) \times (H - f + 1) \times F \tag{2.55}$$

The term F refers to the number of applied filters.

2.4.2 Pooling layers

The role of the *Pooling layer* is to reduce the spacial size of the input image by decreasing the number of parameters and the computational cost. The pooling neurons have not weights and thought an aggregation function, max o average, reduce the input dimensions. The pooling method is applied independently on every channel. This layer is usually applied after a convolutional layer with ReLU activation function. There are two type of pooling layer: *Max pooling* and *Average pooling.*

Max pooling At first the input image is divided in areas, usually non overlapping each other, with dimension equal to the one of the pooling layer. Then is applied a max filter on them. The max filter outputs the maximum value contained in each subarea. This technique highlights the feature contained in the image.

Average pooling As the max polling the image is divided into subareas and an average filter is applied. The average filter preserves also the less important features collapsing the representation without losing information of it.

Figure 2.23 shows the difference between the application of a Max and an Average pooling layer to the same image. The input image has size 4×4, the pooling layers have size 2×2 and a stride of 2 is applied.

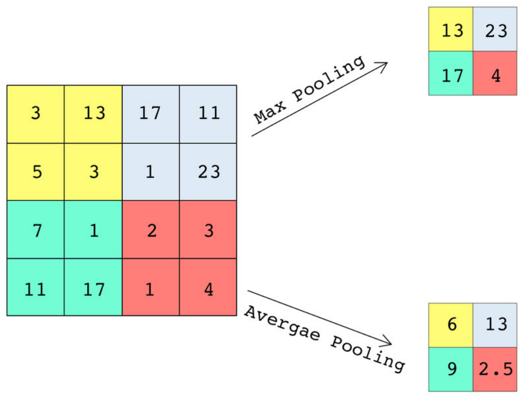


Figure 2.23: Comparison between Max and Avg Pooling.

2.4.3 Convolutional Neural Network Architecture

A convolutional layer can be composed by multiple kernels and can give as output the corresponding pack of images called feature maps. Each feature of the kernel convolute with each channel of the input image.
The most general shape of a CNN input is $B \times W \times H \times F$. The term B refers to the batch size, W to the width, H to the height and F to the number of applied filters.
Usually a CNN is composed by some convolutional layers stacked together alternated by a pooling layer and, at the top of the network, are present some fully connected layers which output the extracted features.
To better understand the architecture of a CNN, it is examined one of the most known architecture, the LeNet-5. It was created for the recognition of hand-write digits and is based on the MINIST database. The architecture of LeNet-5 is the following.

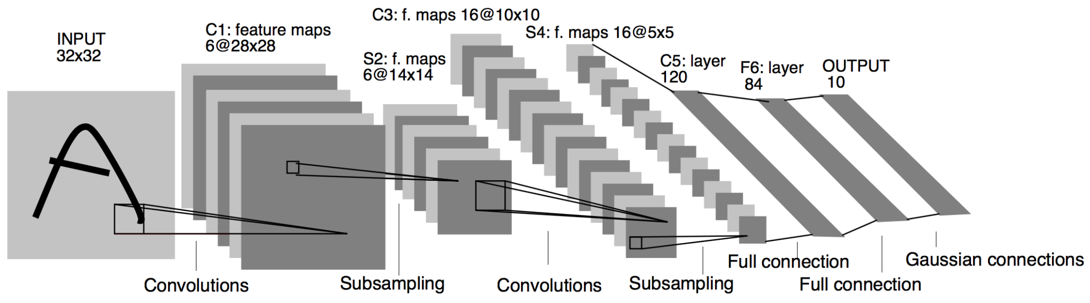


Figure 2.24: LeNet-5 architecture.

It is composed by:

- Input image of shape 32 x 32;
- Six convolutional layers of shape 5x5 with stride s=1;
- Average pooling layer of shape 2x2 with stride s=2;
- Six convolutional layers of shape 5x5 with stride s=1;
- Average pooling layer of shape 2x2 with stride s=2;
- Two fully connected layers of 120 and 84 neurons respectively;
- A Softmax layer which outputs ten different classes.

Can be notice another important detail, the number of channels increase toward the top of the network while the width and the height decrease.

Chapter 3

Methodology

In this chapter is described in details the general structure of the project analyzing the pipeline, the network models, the hardware and software components.

3.1 Project description

The goal of this project is the developing of an algorithm able to detect the pose of a person and recognize the action done. The actions that can be recognized are three; standing, sitting and lying. Several versions of the project have been developed utilizing different datasets.

The first step of the process is the acquisition of an image through the Intel RealSense Depth Camera D435i. The input image has dimension 480×640 pixels and, as soon as it is acquired, it is feed to the PoseNet algorithm.
PoseNet is a machine learning model that allows the real time pose estimation thought the detection of seventeen points. These points correspond to the most important human joints: nose, left eye, right eye, left ear, right ear, left shoulder, right shoulder, left elbow, right elbow, left wrist, right wrist, left hip, right hip, left knee, right knee, left ankle and right ankle.
The pose estimation is run on a TPU device to obtain better performances and maintain an high frame rate, that is around the 30 FPS.
Then, the joint positions and scores are organized in a vector (MLP model) or in a matrix (CNN model) and ,using the previously trained model, is computed a prediction of the pose. The classification label result is showed thought a label on the screen and it is updated in real time.

3.2 Project pipeline

In this section is described the structure of the project pipeline.

Libraries and paths import

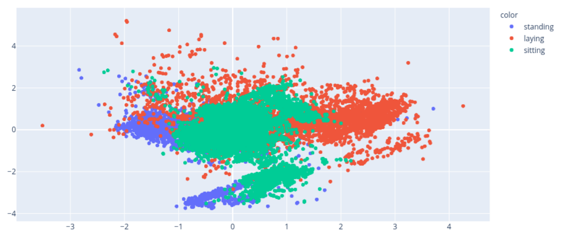
The first operation is importing the libraries and paths that will be used in the development of the algorithm. In addition to the standard libraries as *numpy*, *opencv*, *matplotlib*, etc, are also imported *sklearn* and *keras*.
Sklearn is a Machine Learning library that allows, in a simply way, to work with supervised and unsupervised learning. It helps also in data preprocessing, model fitting, selection and evaluation thought specific tools.
A particular class of the Sklearn library can be highlighted, the *PCA*. Principal Component Analysis, or PCA, is a dimensionality-reduction method used to reduce the number of variables of a dataset. This process lets to represent multidimensional datasets in 2D space increasing the interpretability, minimizing the information loss and allowing to visualize the datasets by means of a graph.
Another important library is Keras, an high level API of TensorFlow 2. It provides building and abstraction blocks to foster machine learning solution in a small amount of time.

Data set import

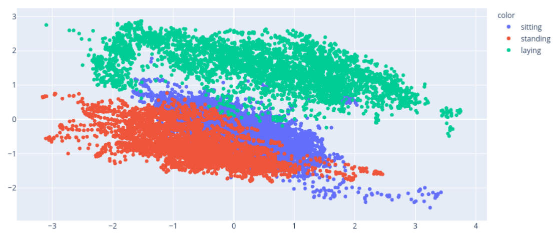
The dataset, saved as pickle format, is imported and divided in two part. The input vector X contains the joint positions and scores while the Y vector contains the action labels.
To work properly, the Y label vector is *one-hot* encoded. The one-hot encoding codifies the text label into a vector, of shape (1,3), composed by 0 and 1. The three actions, standing, sitting and lying are coded as [0,0,1], [0,1,0], [1,0,0] respectively.

Data set representation

As previously introduced, PCA is a dimensionality-reduction method used to increase the interpretability and visualize the data by mean of a graph.
This method has been utilized to visualize the datasets employed in the training of the MLP models. Two MLP models have been built using two different datasets, one without taking the information of the joint's depth and the other with. The reduction of the datasets dimensions are from 51 to 2, in the fist model, and from 68 to 2 in the second one.
In figure 3.1 is represented the data distribution of the two datasets. Sub-figure 3.1a refers to the dataset containing the depth information while sub-figure 3.1b the dataset not containing the depth information.



(a) Representation of the datasets containing depth information



(b) Representation of the datasets not containing depth information

Figure 3.1: PCA representation of two different datasets.

The different colors refers to the three different poses that the algorithm is able to recognize.
To check if the dataset can be used to train the model achieving good results in the pose recognition, it is needed to analyse the points distribution.
If the points of different colors forms a well distinguished could, the probability that, after the training phase, the algorithm is going to predict the right pose is high. This implies that the classes are disjointed and recognizable.
As can be seen some points will overlap, this means that the joint information are similar and the prediction of the correct pose can be not achieved.

Data set preprocessing

The X and Y vectors are partitioned into *Train*, *Test* and *Validation* sets. The Train set constitute the 80% of the whole data set and the Test set the 20%. The Test set is then derived partitioning the 10% of the Validation set. The obtained sets, X_train, X_test, X_val, are normalized using the *min-max normalization* method.
The min-max normalization method consists in transforming all the values scaling them by the same quantity, so each feature has the same weight. The minimum value of each feature is transformed into 0 and the maximum into 1. As consequence all the intermediate values range from 0 to 1. The normalization is done to compare features that have different scales, as for example the score and the depth of a joint.

Model building

The build of the model is one of the most import operation since is defining the structure and the type of the network. The structure used to build the MLPs and CNN models is the *Sequential* one.
The Sequential model is composed by a single stack of sequentially connected layers, where each input and each output has only one tensor. In this project are exploited two different network's models: the Multi Layer Perceptron and the Convolutional Neural Network.

Model training

After the building phase the model is ready to be trained.
The training is a crucial step since all the interconnection weights are optimized to minimize the loss function. To train the model the normalized training sets, X_train and y_train, are feed to the algorithm. The batch size and the number of epochs are set and the validation set is chosen. This approach is called *supervised training*, since both input and output are provided.
The training process is simple: the output, obtained processing the inputs through the network, is compared with the desired output. The resulting error is backpropagated and the weights are tuned. Those operations are repeated until the system has been correctly trained.
The obtained model is then saved and can be subsequently load.

To help the training of the model, the loss and the accuracy of both Training and Validation set can be visualized. From the derived graph can be checked the model *underfitting* or *overfitting*.
Overfitting refers to a model that represents the training data too well. Underfitting, on the contrary, refers to a model that is not able to fit the training data. The figure 3.2 shows the relation between the training and validation loss over the time, compared with the overfitting and underfitting problem.

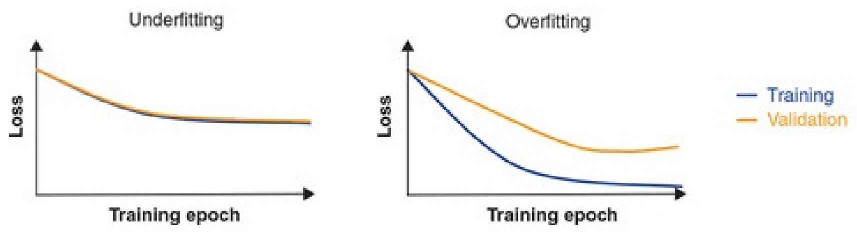


Figure 3.2: Overfitting vs Underfitting problem

Model testing

Before working on the real project the model have been tested to check its performances.
The test phase consist on passing the normalized test set X_test to the *evaluate* function in order to estimate the performances of the previously trained model. The outputs are the loss and accuracy of the model.

Model prediction

The Keras function *predict*, estimates a result for each entry of the given set. The input set is the normalized test set X_test and the output is a vector containing the one encoding of the estimated pose. To evaluate the accuracy of the classification model has been computed the *Confusion Matrix*.
The Confusion Matrix is a matrix which diagonal elements refer to the number of points for which the predicted label is equal to the actual one. The off-diagonal elements refer to points that are not correct labeled. Analyzing this matrix, the information about the accuracy of the model can be retrieved and, if the expected performances are not satisfied, the model can be retrained.

PoseNet and Pose prediction

Once the model has been trained and tested to achieve the desired performance, it is integrated in the classification algorithm. The classification algorithm can be divided in two parts, PoseNet and classification.
The RGB and depth images, acquired by the RealSense camera, are aligned to have a one to one correspondence between the pixels of the image and the depth map entries. The RGB image is then feed to the PoseNet algorithm.
The PoseNet algorithm returns the score of the pose, the scores and the positions of the joints relative to that pose. If the score of the pose is above a certain threshold it is accepted as valid and a vector containing the keypoints information is built.
The depth information about the keypoints are obtained using the acquired depth maps and the x,y joints position. This is done by taking the value inside the cell of the depth map located in the (x,y) position. This operation is repeated for each of the 17 keypoints. The previously created vector is then updated with the depth values.
Then, the obtained vector has been normalized. The classification function is executed, the resulting output is transformed ,for the sake of simplicity, from one-hot encoding to a single value and it is checked a correspondence with one of the expected classes. The recognized pose is showed on the screen.

Those operations are real-time executed to have a continuous tracking of the observed person pose.

3.3 Loss function

The loss function measures the learning quality of the algorithm. It is calculated and updated constantly during the training of the network.
The loss function, or cost function, is computed by calculating the error between the actual output $\hat{y}$ and the desired output y. This quantity must be minimized to obtain a minimum variation between the desired and the actual output.

The cost function used by the models is the *Categorical Cross Entropy*. The Categorical Cross Entropy, also called Softmax loss, is composed by a Softmax followed by a Cross Entropy loss function. This function has been selected since is widely used in multi-classification problems where the predicted output can belongs only to one class.

3.4 Optimization algorithm

Optimizers are loss reduction algorithm used to tune the neural network parameters as weights, biases and learning rate. These methods provide accurate results updating the parameters toward the minimum direction.
The optimization algorithm that has been used is *Adam*. Adam algorithm, or Adaptive Momentum Estimation algorithm, is an adaptive optimizer. The adaptive optimizers can constantly adapt the parameters without waiting the beginning of the training process. Adam optimizer is a combination of RMSprop and Momentum method.
At first, the exponentially weighted average of the past gradients is calculated. Then, the exponentially weighted average of the squares of the past gradients is computed. Finally, information retrieved from the previous computations are used to update the parameters. Adam algorithm is chosen as default method when a neural network is trained due to its high performances.

3.5 Metrics

The choice of the correct metric is a fundamental aspect in the evaluation of a neural network model. Metrics are used to supervise and measure the performance of a model.
Int his section four of the most important classification related metrics are analyzed:

- Confusion Matrix
- Accuracy
- Precision

- Recall

Confusion Matrix Although the *Confusion Matrix* is not considered a metric, it is one of the fundamental concept in checking the classification performances of a model.
It is a tabular representation of the model prediction versus the ground-truth labels. The instances in a predicted class are represented by the confusion matrix rows, instead the instances in an actual class are related to the confusion matrix columns. The diagonal elements contains the correct predicted label and the off-diagonal one refer to instances that are not correct labeled.

Accuracy *Accuracy* is the most simple metric, it is defined as the number of correct predictions divided by the total number of predictions. To obtain the measure in percentage it must be multiplied by 100.

$$\text{Classification accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}} \tag{3.1}$$

The accuracy metric not performs well in case of unbalanced class distribution. This means that the probability to predict one class is higher than then others. To avoid wrong evaluation of the model a new method has to be introduced.

Precision *Precision* metric is used to overcome the unbalanced class distribution. It is defined as:

$$\text{Precision} = \frac{\text{True positive}}{(\text{True positive} + \text{False positive})} \tag{3.2}$$

Recall Another essential metric is the *Recall*.
It is determined as the fraction between the correctly predicted elements and the total number of samples.

$$\text{Recall} = \frac{\text{True positive}}{(\text{True positive} + \text{False negative})} \tag{3.3}$$

3.6 Hardware and Software setup

In this section is analyzed the working platform and it is explained the Hardware and the Software setup that has been utilized.

3.6.1 Hardware setup

The requested hardware for the development of the thesis has been provided by the PIC4SeR Centre at Politecnico di Torino.
The hardware components used to develop this project are:

- Intel Ralsense Depth Camera D435i
- Google Coral USB Accelerator, Edge TPU coprocessor

Intel Realsense Depth Camera D435i

The Realsense Depth Camera is a stereo-camera able to provide high quality RGB and depth images. It offers complete depth cameras integrating vision processor D4, stereo depth module, RGB sensors with color image signal processing and a IMU.[10]
This camera has a very wide range of application and, as said by Intel, "the combination of a wide field of view and global shutter sensor on the D435i make it the preferred solution for applications such as robotic navigation and object recognition."[11]
The D435i camera is composed by a RGB module, two infrared cameras and a IR projector. The detailed hardware components of the camera are showed in the figure below.

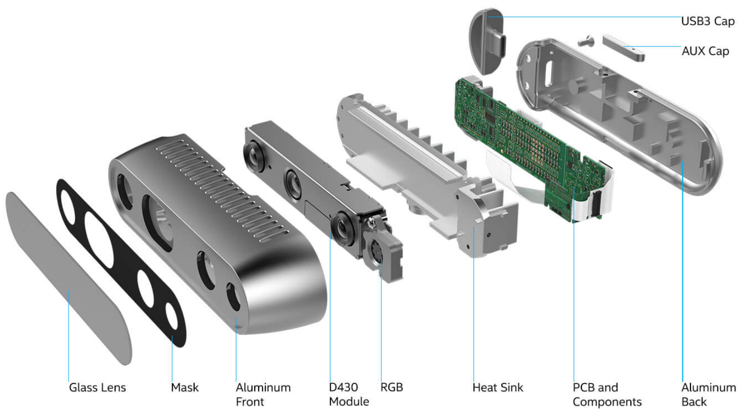


Figure 3.3: Detailed Realsense Depth Camera D435i hardware components.[11]

The types of camera that are available using the Realsense Depth Camera D435i are:

- RGB, 1920x1080 max resolution
- Depth, 1280x720 max resolution

- Infrared 1
- Infrared 2

Google Coral USB Accelerator

The Coral USB Accelerator adds an Edge TPU coprocessor to the system, enabling high-speed machine learning inferencing on a wide range of systems.[12]
The TPU is a device based on the ASIC technology, created by Google, for neural network machine learning applications. The coprocessor mounted on the Edge TPU can perform a huge amount of operations per seconds. It can be easily interfaced thought USB to a lot of devices and operating systems. An important aspect is that TensorFlow Lite models can be compiled to run on it.

In this thesis the USB Accelerator has been used to run the PoseNet model allowing the recognition of the person pose in real-time. The usage of this TPU device reduces the computational effort done by the CPU of the computer.

3.6.2 Software setup

To develop a machine learning project a lot of programming languages can be used but in this specific case was chosen *Python*.
Python is the most used programming language for Machine Learning, it is well supported and has a lot of built-in libraries.
To build this project was used the release 3.8 of Python and the following libraries:

- numpy
- pandas
- keras
- sklearn
- matplotlib.pyplot
- plotly.express
- PIL
- cv2
- pyrealsense2

The whole code is written in Python on a web-based environment called *Jupyter notebook*.
The Jupyter notebook is an open-source web application that allows to create documents with code and other languages as markdown. The document can be divided in module, each module can be executed independently and can be run via a web browser. The kernel used in this project is the *ipyhon* kernel that executes python code but many other kernels are available.

Tensorflow and Keras Tensorflow is an end to end, open-source machine learning platform. [13]
It provides APIs for Python and other programming languages and is widely used for Machine Learning applications.
The TensorFlow library is specifically designed to build, train and test machine learning models providing modules, classes and functions to the network designer. Keras is the high-level API of TensorFlow which provides building blocks for implementing machine learning and deep learning models. It minimizes the action required by the end user speeding up the model creation process.
As TensorFlow, Keras maintains the scalability and cross-platform features and can be run on TPU or GPUs, can be exported in different format or it can be run in the browser, on mobile or embedded devices.

Chapter 4

Experimentation

In this section the steps done to implementation the project described, from the creation of the dataset to the design of the neural network model.

4.1 Dataset creation

The necessity of the creation of a dataset derives from the fact that during the research phase no valid datasets are found.
The project has been intended to recognize and predict the pose of a single person using a wheeled robot. The camera is supposed to be mounted on the robot at an height of approximately twenty centimetres which implies that the tracked person is shot from above. Due to this characteristics the existing datasets are not valid since in most of cases there are multiple people and the frame is taken by a camera mounted on the ceiling. The first step of this thesis has been the creation of a valid dataset that satisfies the problem requirements.

4.1.1 Image acquisition

Photo camera

The first dataset is implemented using a simple photo camera, precisely a Canon 77D, mounting a wide-angle lens to reproduce the angle of the RGB and Depth camera mounted on the robot. To acquire the photographs a remote controller is used.
The resolution of the images acquired by the camera is 1600×2400 and since PoseNet requires as input a dimension of 480×640 the images are cropped. The resizing cannot be done since it deforms the image ratio influencing the relative position of the joints during the pose estimation. This can negatively affect also the training and recognition steps. The total number of collected images are 780

subdivided in three category: standing, sitting and lying, containing respectively 250, 350 and 180 images.

Intel Realsense Depth Camera D435i

Thanks to the first dataset, is found out that the action recognition is quite accurate. Then, a new dataset is created. The image acquisition is done using the camera that will be employed in the real time pose estimation, the Intel RealSense Depth Camera D435i. The use of this camera allows to acquire the depth map of the image and to exclude problems related to the different kind of images used for the training of the model and for the pose estimation.

RGB Image acquisition The image acquisition using the RealSense camera is done in two steps.
At first some videos of standing, sitting and lying person are shot. Then, each frame of the videos is extrapolated and saved as a image. The size of the obtained images is 480×640. Finally, the images can be fed directly to the PoseNet algorithm. The total number of images used for this dataset is 25009. The images are divided in three classes, standing, sitting and lying containing 7849, 11400, 5760 images respectively.

Depth map acquisition The acquisition of the depth map is related to the acquisition of RGB images.
While the RGB videos are shot also the videos representing the depth map are saved. The next operation is the extrapolation of the frames from the videos. The result is a matrix with shape 480×640 for each image. The matrix contains the depth measure of each pixel that in the pre-processing step are used to acquire the z position of the seventeen joints.

Data acquisition for CNN model

To train the CNN model are no more used vector but matrices of shape 480×640×2. The images and the depth maps used to build the matrices are the one acquired for the creation of the RGB and depth dataset. Although the images and the depth maps are the same the main difference between the datasets is the pre-processing step.

4.1.2 Pre-processing

Once the images and the depth maps are collected they are pre-processed to obtain several datasets which are used to train the different neural network models.

Photo camera

At first, each image is fed to the PoseNet algorithm to obtain the x, y position and the scores of the seventeen joints. Then, a *one-hot* encoded vector, which refer to the pose detected in the image, is created. Finally, the information obtained from the first step, the one-hot vector and the name of the image are stacked together in a vector and saved as pickle format. The structure of a generic vector is the following:

$$[x_0, \cdots, x_{16}, y_0, \cdots, y_{16}, s_0, \cdots, s_{16}, [\textit{one-hot label}], '\textit{image\_name}'] \tag{4.1}$$

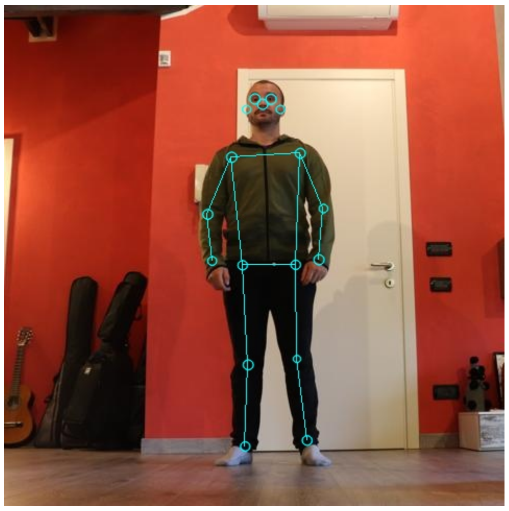
where:

- $[x_0, \cdots, x_{16}]$ refers to the x positions of the 17 joints recognized by PoseNet;
- $[y_0, \cdots, y_{16}]$ refers to the y positions of the 17 joints recognized by PoseNet;
- $[s_0, \cdots, s_{16}]$ refers to the scores of the 17 joints recognized by PoseNet;
- [*one-hot label*] refers to the pose name transformed in one-hot encoding. [0 0 1] stands for standing, [0 1 0] stands for sitting and [1 0 0] stands for lying.

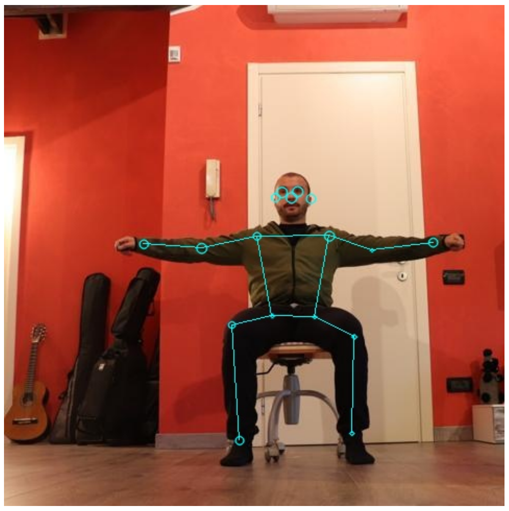
The vectors obtained thought this process are then stacked together as columns obtaining the whole dataset. This first dataset has still few images but has been used to do the first tests on the network to verify the actual behaviour of the system.

The following images show the human body keypoints detection using PoseNet algorithm.

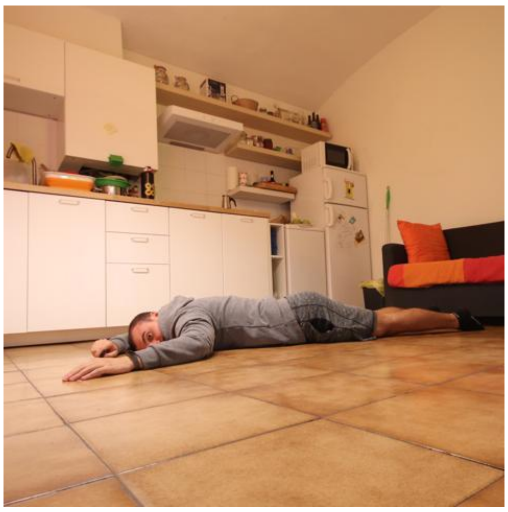
(a) Original acquisition


(b) Processed with PoseNet

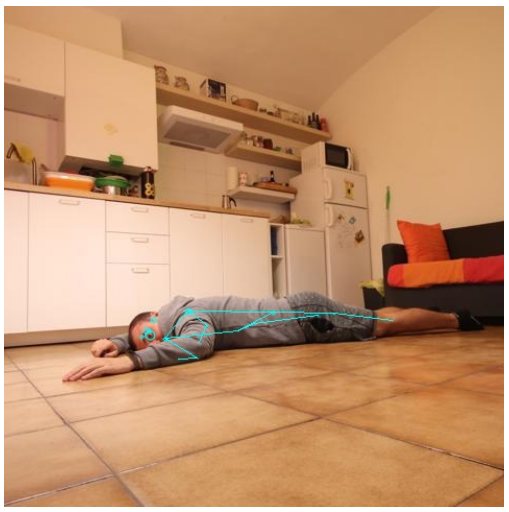
(c) Original acquisition



(d) Processed with PoseNet



(e) Original acquisition



(f) Processed with PoseNet

Figure 4.1: Comparison between original images and processed ones using PoseNet algorithm.

RGB and Depth camera

The pre-processing step for the RGB and Depth camera is the same as the one considered for the photo camera since the only difference is the device used for the image acquisition. Instead, for the case in which the depth is considered, the pre-processing step requires a slightly different approach.
To the previously described vector is added the depth measure of each keypoint.

A program take as input the RGB image and the corresponding depth map. The joint position is obtained using PoseNet, which extrapolate the x,y position. This position has been used to retrieve the content of the cell (x,y) of the depth map. The value inside the cell in the depth measure of the keypoint. This process is repeated for all the joints obtaining the depth vector. The depth vector $[z_0, \cdots, z_{16}]$ is added to the existing data to derive the complete dataset structure.
The generic vector of the dataset is:

$$[x_0, \cdots, x_{16}, y_0, \cdots, y_{16}, s_0, \cdots, s_{16}, z_0, \cdots, z_{16}, [\textit{one-hot label}], '\textit{image\_name}'] \tag{4.2}$$

CNN matrix

The CNN to be trained requires a different data structure. The structure of a generic entry of the dataset is 480×640×2. It is a 2D matrix which contains different information per layer.
The steps to build the matrix are the following:

- the matrix is initially set to zero;
- the first layer, in correspondence to the x,y joint position is filled with the relative keypoint score;
- the second layer, in correspondence of the x,y joint position is filled with the relative keypoint depth.

The image acquisition and the pre-processing step used to build this dataset are the same used to build the RGB and Depth one since the information retrieved from the images and the depth maps are the same. The main difference between the two datasets is the organization of the data.

4.2 Neural Networks Architectures

In this section are analyzed the Neural Network models used to implement the project.
Two model architectures have been used: Multi Layer Perceptron and Convolutional Neural Network.
The goal of this project is the development of a pose recognition and classification algorithm capable to recognize the pose of a person. The whole system has to be mounted on a wheeled robot and the recognition step need to be done in real time. This lead to a trade-off between precision and computational efficiency.
The models that have been developed need to be as simple as possible to speed up the computations. To do this the model complexity is reduced until good performance are obtained.

Two different model architectures are used: *Multi Layer Perceptron* (MLP) and *Convolutional Neural Network* (CNN).

4.2.1 MLP model

The Multi Layer Perceptron is formed by three different kind of layers: one *input* layer, one or more *hidden* layers and one *output* layer. The hidden layers are formed by Linear Threshold Units (LTU) and are not visible from the outside.
The MLP model is used to implement two different version of the algorithm. The versions differ from the use of the depth information of the joints, which implies the use of two distinguished datasets.
The type of layers that have been used is the *Dense* layer.
The Dense layer, also called *non-linear* layer, is a fully connected layer and is widely used in neural networks due to its simplicity.
It is defined by the formula:

$$y = f(wx + b) \tag{4.3}$$

The terms w and b refer to the weights and bias vector, whereas f refers to the activation function.
The architecture of the two model is the same and can be summarized as:

- **Dense** layer with N neurons. **ReLU** activation function;
- **Dense** layer with N neurons. **ReLU** activation function;
- Output **Dense** layer with 3 neurons. **SoftMax** activation function.

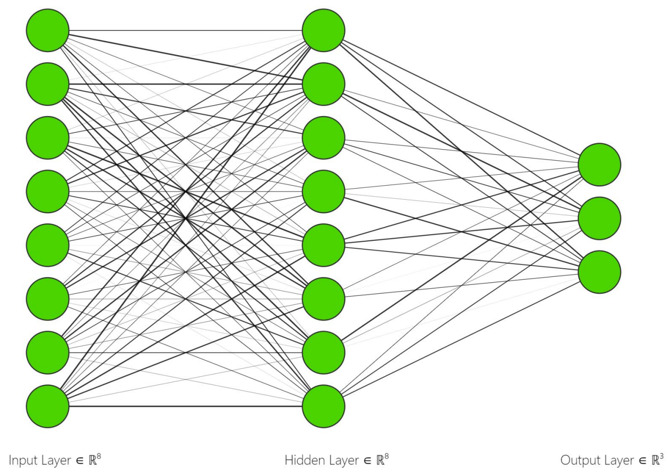


Figure 4.2: Example of a MLP architecture having two Dense layers with 8 neurons each and an Output layer with 3 neurons.

The only difference between the two structures is the number of neurons present in the hidden layers since each model performs under different assumptions.

MLP model without depth

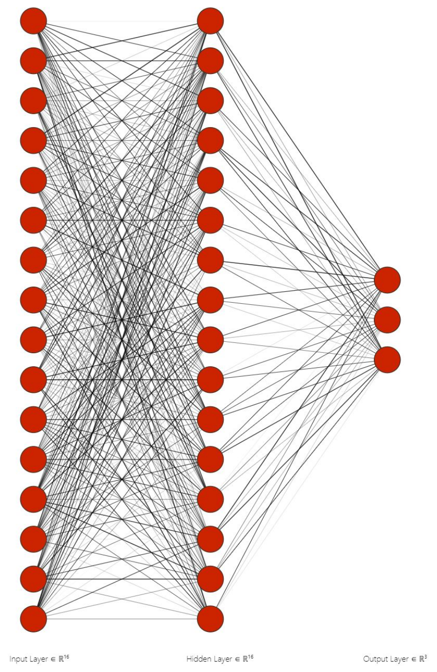
After some tests to tune the parameters and the number of neurons in each layer to achieve good performances, the architecture of the model is:

- **Dense** layer with 16 neurons. **ReLU** activation function;
- **Dense** layer with 16 neurons. **ReLU** activation function;
- Output **Dense** layer with 3 neurons. **SoftMax** activation function.

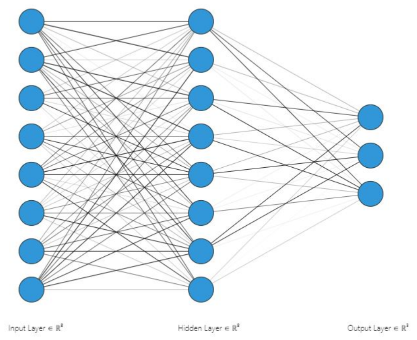
MLP model with depth

After some tests to tune the parameters and the number of neurons in each layer to achieve good performances, the architecture of the model is:

- **Dense** layer with 8 neurons. **ReLU** activation function;
- **Dense** layer with 8 neurons. **ReLU** activation function;
- Output **Dense** layer with 3 neurons. **SoftMax** activation function.


Input Layer ∈ $\mathbb{R}^{16}$
Hidden Layer ∈ $\mathbb{R}^{16}$
Output Layer ∈ $\mathbb{R}^{3}$

(a) MLP model without keypoints depth information


Input Layer ∈ $\mathbb{R}^{8}$
Hidden Layer ∈ $\mathbb{R}^{8}$
Output Layer ∈ $\mathbb{R}^{3}$

(b) MLP model with keypoints depth information

Figure 4.3: Architectures of the MLP models.

4.2.2 CNN model

The architecture of a CNN is usually composed by: some convolutional layers stacked together alternated by a pooling layer and, at the top of the network, some fully connected layers which output the extracted features. The detailed architecture constituting the model is reported below.

- Conv2D layer 32,3; stride of 2; ReLU activation function;
- Conv2D layer 32,3; stride of 2; ReLU activation function;
- Conv2D layer 32,3; stride of 2; ReLU activation function;
- Average pooling 2D layer; pool size 2; stride 2; padding valid;
- Flatten layer;
- Dense 3, Softmax.

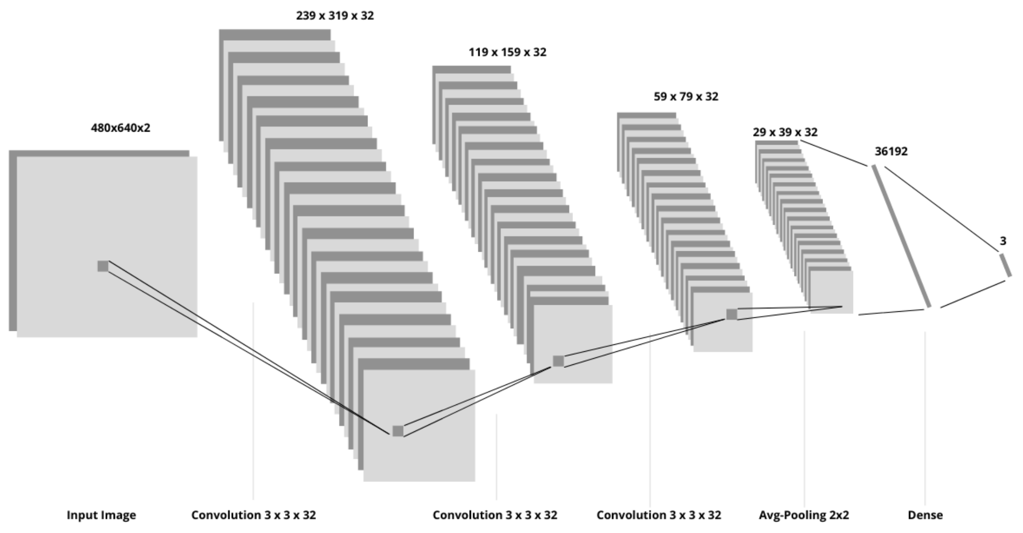


Figure 4.4: Architecture of the CNN model.

4.2.3 Common elements

Despite the two models, MLP and CNN, have several differences there are some elements in the network architecture that not change.

Loss function The cost function used by the models is the *Categorical Cross Entropy*. The Categorical Cross Entropy, also called *Softmax loss*, is composed by a Softmax followed by a Cross Entropy loss function. This function has been selected since is widely used in multi-classification problems where the predicted output can belongs only to one class.

Optimization algorithm The optimization algorithm that has been used is *Adam*. Adam algorithm, or Adaptive Momentum Estimation algorithm, is an adaptive optimizer.
The adaptive optimizers can constantly adapt the parameters without waiting the beginning of the training process. Adam optimizer is a combination of RMSprop and Momentum method. At first, the exponentially weighted average of the past gradients is calculated. Then, the exponentially weighted average of the squares of the past gradients is computed. Finally, information retrieved from the previous computations are used to update the parameters.
Adam algorithm is choose as default method when a neural network is trained due to its high performances.

Metrics Metrics are used to supervise and measure the performance of a model. Two metrics are used to check the models performances, Confusion Matrix and Accuracy.

Confusion Matrix The Confusion Matrix is a tabular representation of the model prediction versus the ground-truth labels. The instances in a predicted class are represented by the confusion matrix rows, instead the instances in an actual class are related to the confusion matrix columns. The diagonal elements contains the correct predicted label and the off-diagonal one refer to instances that are not correct labeled.

Accuracy The accuracy is defined as the number of correct predictions divided by the total number of predictions. To obtain the measure in percentage it must be multiplied by 100.

$$\text{Classification accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}} \tag{4.4}$$

Chapter 5

Results and Conclusions

In this section the results about the three different models are presented and a discussion about the future implementations is done.

The goal of this project is the developing of an algorithm able to detect the pose of a person and recognize the static action done. The actions that can be recognized are three: standing, sitting and lying. Three models have been used and for each of them good results have been achieved.
The accuracy obtained by the models is around the 98%. It is a very high accuracy but some considerations need to be done.
The datasets have been created especially for this purpose and no comparison with other similar projects can be done to check the real accuracy of the models. Moreover, the accuracy can be affected by the number of images that constitute the different datasets. The images which have been acquired to train and test the models are few, around 20000, and only two background scenarios have been used. This can lead to prediction problems when the system is used different scenarios. The last criticism is represented by the decrease of the accuracy when the object to be detected is too far from the camera, specially in the case in which are taken into account the depth information of the keypoints. It is an expected behaviour since the datasets are design to detect a person at a maximum distance of two or three meters away from the camera, since the robot on which the system need to be mounted is following the person. Despite these critical aspects good results have been obtained giving a precise pose recognition.

The neural networks models have been developed to be as simple as possible without compromising the fulfillment of the performances. Below are analyzed in details the performances of the different models.
An empirical approach has been used to find the best solution for the MLP and CNN models.

5.1 MLP

The general structure of the MLP models is the following: The optimization algorithm that has been used is *Adam*.

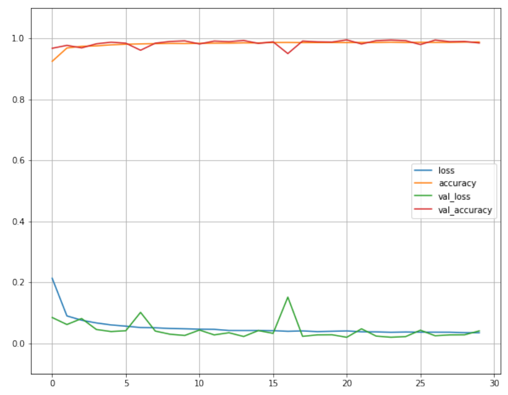
- **Dense** layer with N neurons. **ReLU** activation function;
- **Dense** layer with N neurons. **ReLU** activation function;
- Output **Dense** layer with 3 neurons. **SoftMax** activation function.

This structure remains the same for the two different models since, despite the datasets change, the problem is still a multi-class classification.
The number of neurons per layers are tuned for each of the model to obtain suitable performances.
To have a reference, the models are trained for an equal number of epochs, 30.
The number of neurons per layer is initially set to 4 and then increased until 512.
The obtained results are saved and then compared.
To check the real time results all the versions of the model are run and some videos are acquired to compare the accuracy of the pose recognition algorithm.
Four elements are taken into account to compare the obtained results:

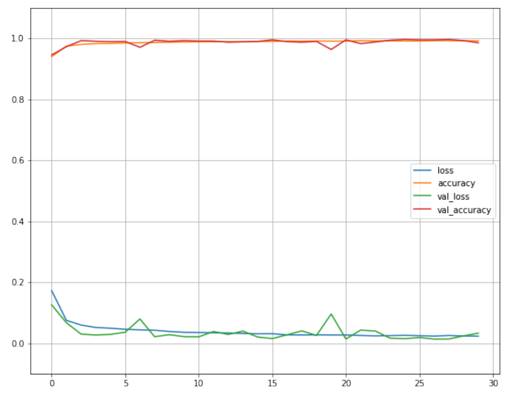
- Loss and Accuracy behaviour of the Train and Validation set over the training epochs;
- Final Loss and Accuracy;
- Confusion Matrix;
- Real time Pose Classification.

5.1.1 MLP without depth

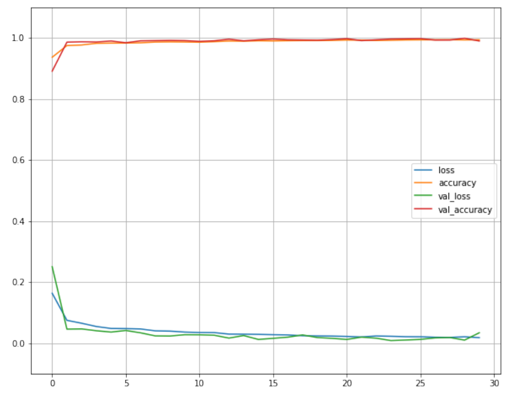
The graphs showing the evolution of the Loss and Accuracy of the Train and Validation set over the training epochs are grouped in figure 5.1. As can be notice during the training of the model, in all the cases, the evolution of the parameters is linear. There are some pikes in the validation loss and accuracy but it is not unusual especially in the early epochs. To overcome this problem the number of training epochs could be increased. Since overfitting or underfitting problems do not occurs and the final results are acceptable the number of epochs is maintained equal to 30.



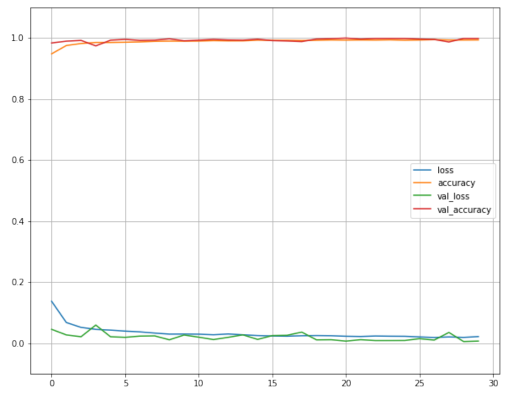
(a) 4 neurons per layer



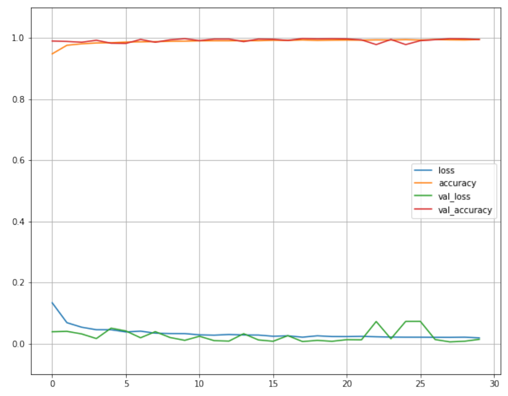
(b) 8 neurons per layer



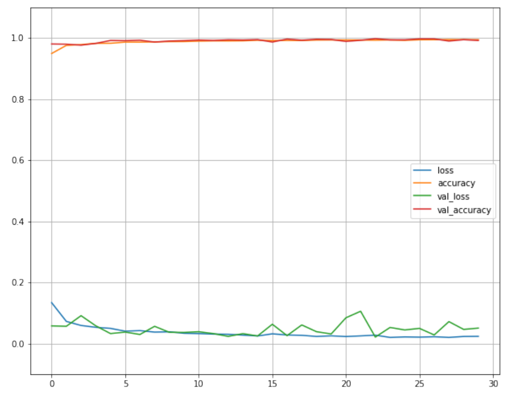
(c) 16 neurons per layer



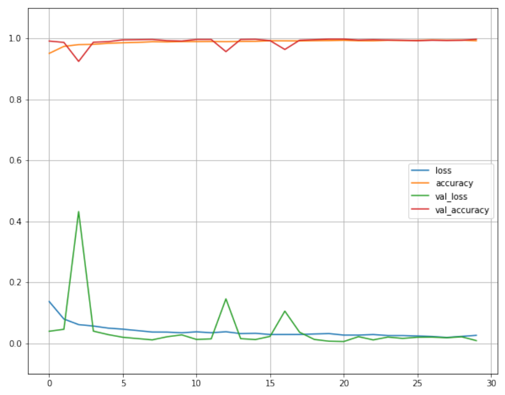
(d) 32 neurons per layer



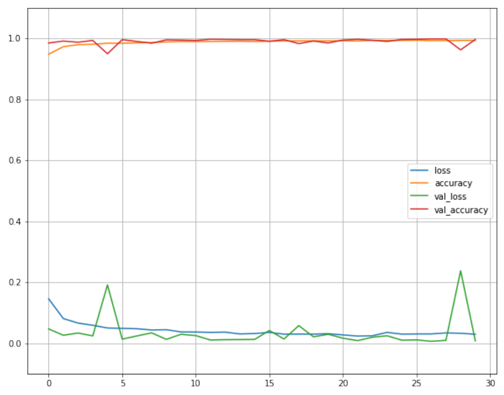
(e) 64 neurons per layer



(f) 128 neurons per layer



(g) 256 neurons per layer



(h) 512 neurons per layer

Figure 5.1: Accuracy and loss behaviour of MLP model.

The obtained results in term of accuracy and loss are reported in the table below.

Number of layers	Accuracy	Loss
4	0.0505	0.9815
8	0.0376	0.9869
16	0.0262	0.9904
32	0.0155	0.9946
64	0.0148	0.9938
128	0.0236	0.9948
256	0.0079	0.9971
512	0.0108	0.9958

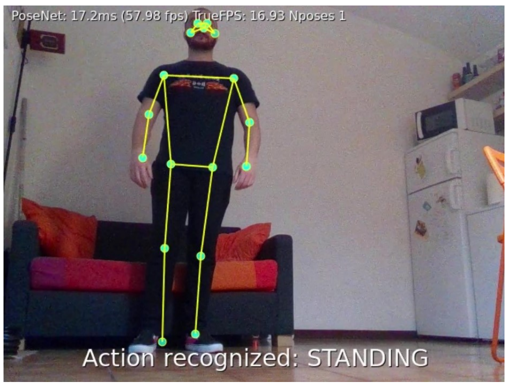
Analyzing the table can be noticed that the best results are obtained with 256 neurons per layer. The structure, considering the optimal number of neurons, can be too complex. The increase of the number of neurons per layer in a such simple network can affect in a negative way the performances. Can be noticed that the accuracy tends to decrease in the last test where the number of layers are 512. The goal is to obtain a model as simple as possible and since the performances do not change in a substantial way, the chosen number of neurons per layer is 16.
The obtained accuracy is around **0.9904** which can be considered as a good result.

To have an idea of the system performances, a prediction using the Test set has been done and the confusion matrix is calculated.

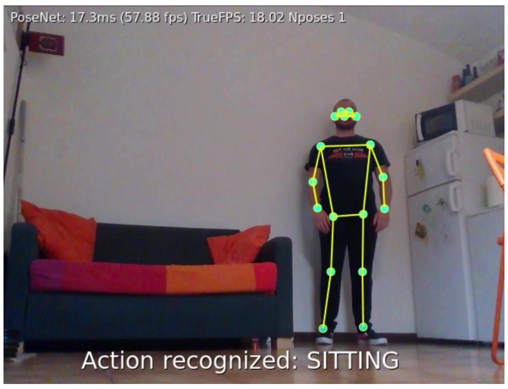
902	0	0
0	1987	2
0	35	965

The wrong predicted poses (off-diagonal elements) are few with respect to the corrected one. In particular the system has some difficulties to distinguish the sitting and the lying pose.

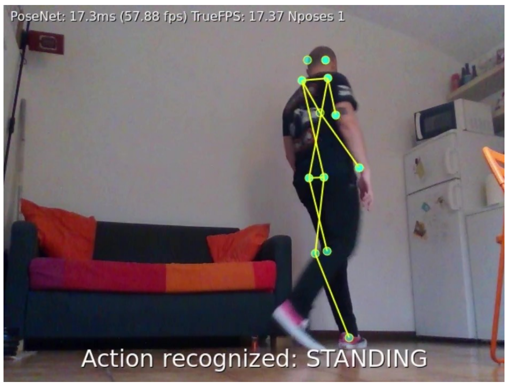
Finally the real time pose recognition algorithm is run and some images are acquired. (5.2)


PoseNet: 17.2ms (57.98 fps) TrueFPS: 16.93 Nposes 1
Action recognized: STANDING

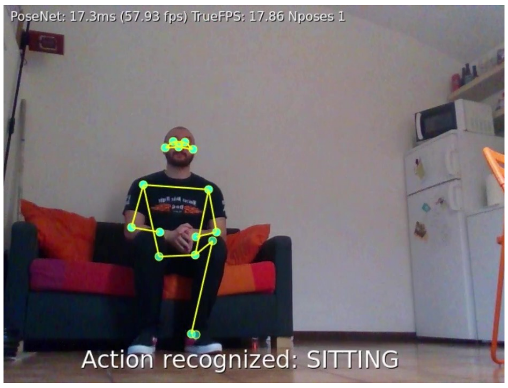
(a) Standing correct


PoseNet: 17.3ms (57.88 fps) TrueFPS: 18.02 Nposes 1
Action recognized: SITTING

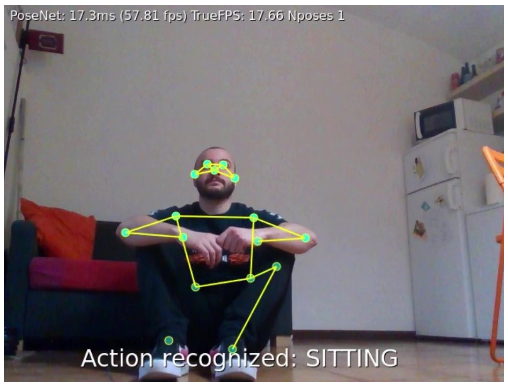
(b) Standing wrong


PoseNet: 17.3ms (57.88 fps) TrueFPS: 17.37 Nposes 1
Action recognized: STANDING

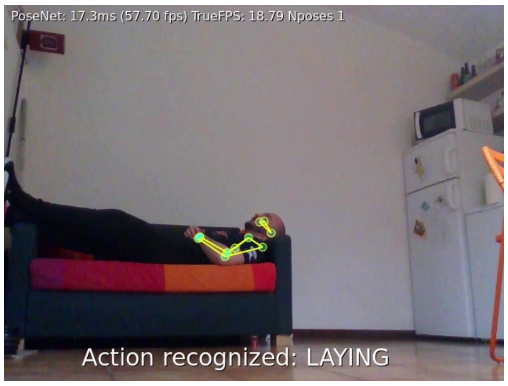
(c) Standing moving


PoseNet: 17.3ms (57.93 fps) TrueFPS: 17.86 Nposes 1
Action recognized: SITTING

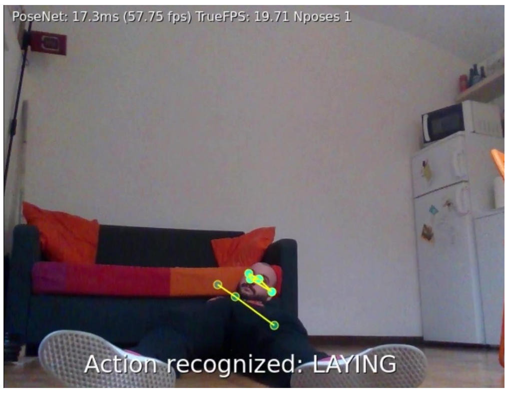
(d) Sitting on a couch


PoseNet: 17.3ms (57.81 fps) TrueFPS: 17.66 Nposes 1
Action recognized: SITTING

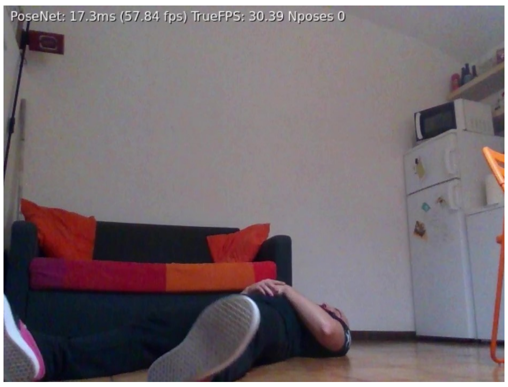
(e) Sitting on the floor


PoseNet: 17.3ms (57.70 fps) TrueFPS: 18.79 Nposes 1
Action recognized: LAYING

(f) lying on a couch


PoseNet: 17.3ms (57.75 fps) TrueFPS: 19.71 Nposes 1
Action recognized: LAYING

(g) lying on the floor



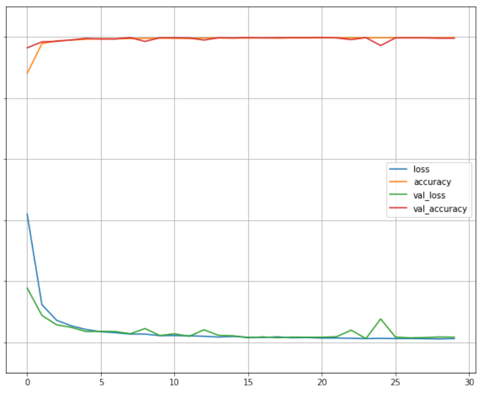
(h) No pose recognized

Figure 5.2: Real time acquisitions of pose recognition algorithm.

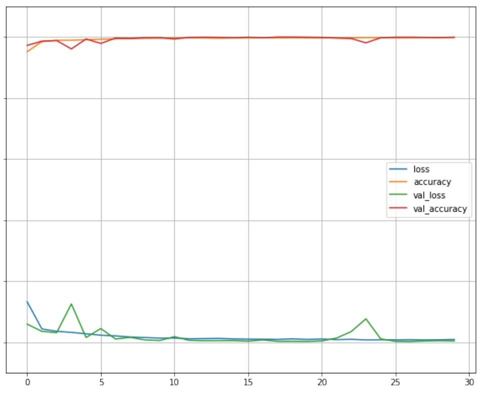
In figure 5.2b is visualized an example of wrong classification. The predicted pose is sitting but the correct one has to be standing. This because if the person to be tracked is too far from the camera the accuracy decreases.
In figure 5.2f can be notice the classification of the lying pose. The classification is correct despite only few keypoints are detected. The recognition of the lying pose is critical due to this aspect.
Finally, in figure 5.2h the algorithm is not able to classify a valid pose since the face keypoints are not detected.

5.1.2 MLP with depth

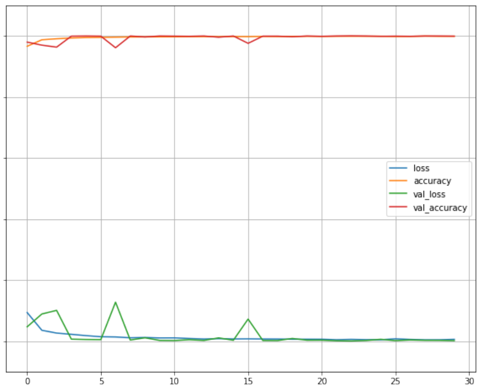
To analyse this model is used the same approach employed previously.
The graphs showing the evolution of the Loss and Accuracy of the Train and Validation set over the training epochs are grouped in figure 5.3. The training epochs are set equal to 30 since no overfitting or underfitting occur.



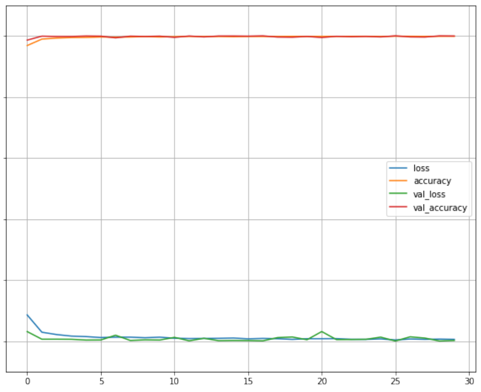
(a) 4 neurons per layer



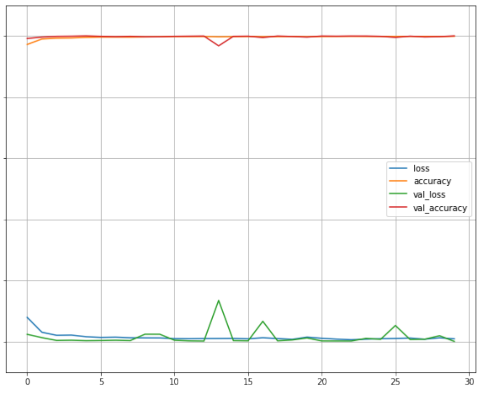
(b) 8 neurons per layer



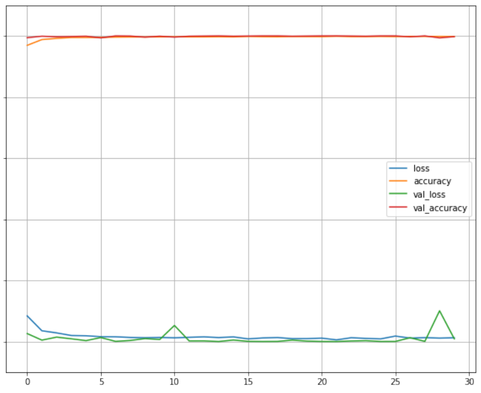
(c) 16 neurons per layer



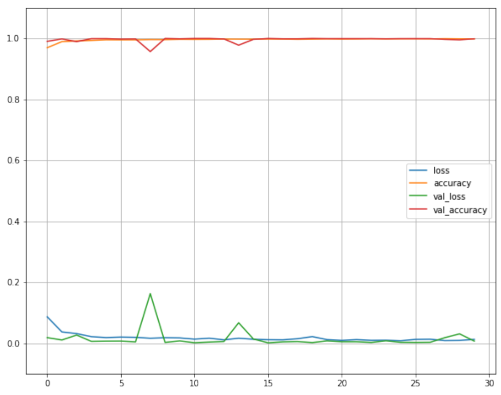
(d) 32 neurons per layer



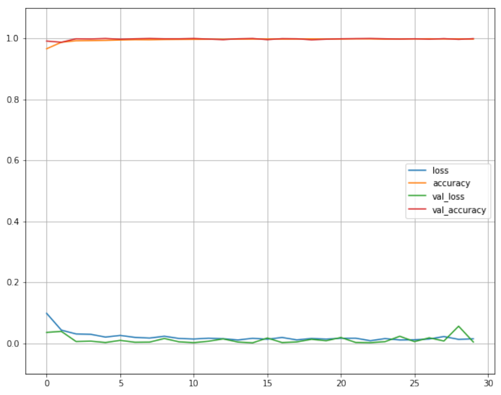
(e) 64 neurons per layer



(f) 128 neurons per layer



(g) 256 neurons per layer



(h) 512 neurons per layer

Figure 5.3: Accuracy and loss behaviour of MLP model considering depth information.

The obtained results in term of accuracy and loss are reported in the table below.

Number of layers	Accuracy	Loss
4	0.0150	0.9972
8	0.0064	0.9984
16	0.0033	0.9988
32	0.0076	0.9982
64	0.0106	0.9990
128	0.0300	0.9910
256	0.0135	0.9988
512	0.0037	0.9988

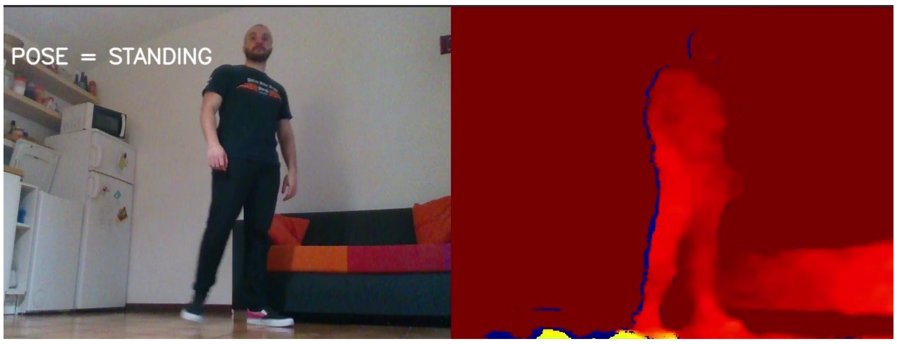
The same reasoning done for the other model can be applied to this one. The best performances are obtained with a number of neurons per layer equal to 64. The model need to be as simple as possible and observing the results good performances are obtained with 8 neurons per layers. This solution is then chosen.
The obtained accuracy is around **0.9984** which can be considered a good result.

To have an idea of the system performances, a prediction using the Test set has been done and the confusion matrix is calculated.

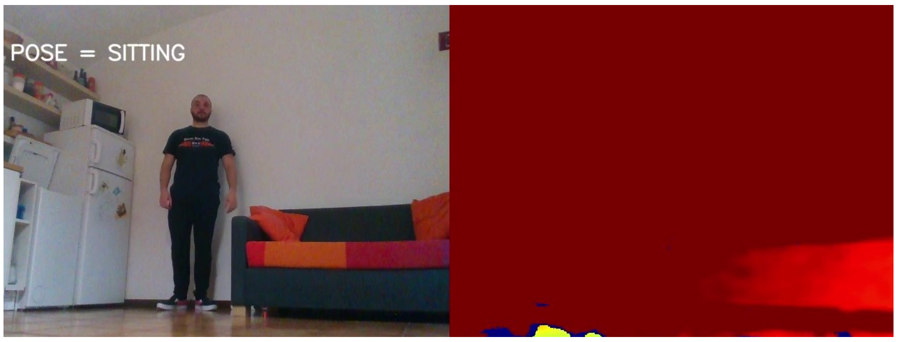
1152	2	0
0	2324	1
0	5	1518

Also in this cases the pose recognition algorithm has some difficulties to distinguish the sitting and the lying pose.

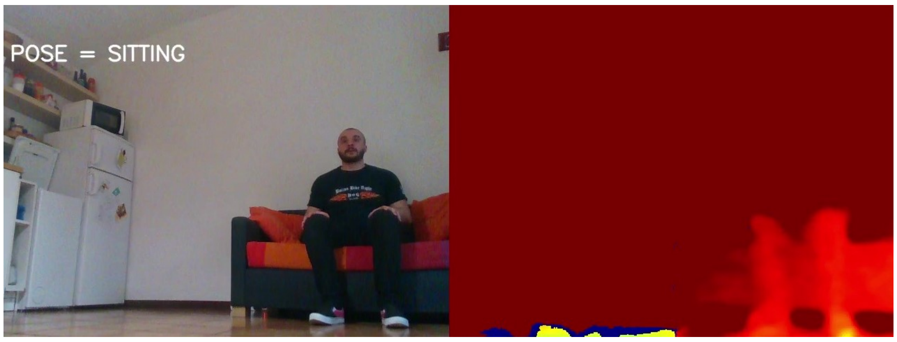
Finally the real time pose recognition algorithm is run and some images are acquired. 5.4



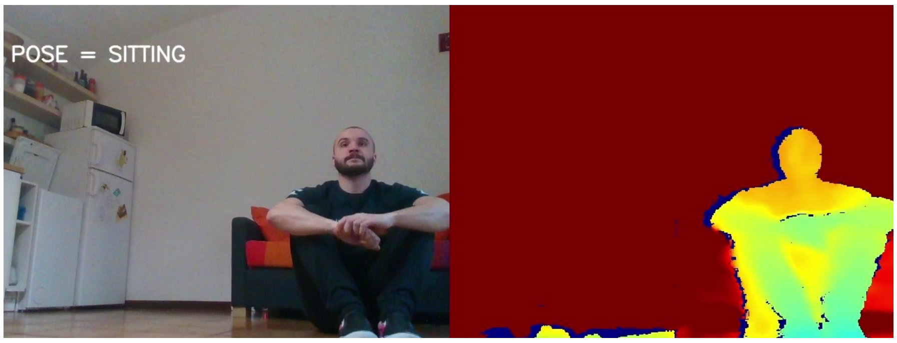
(a) Correct Standing pose recognized



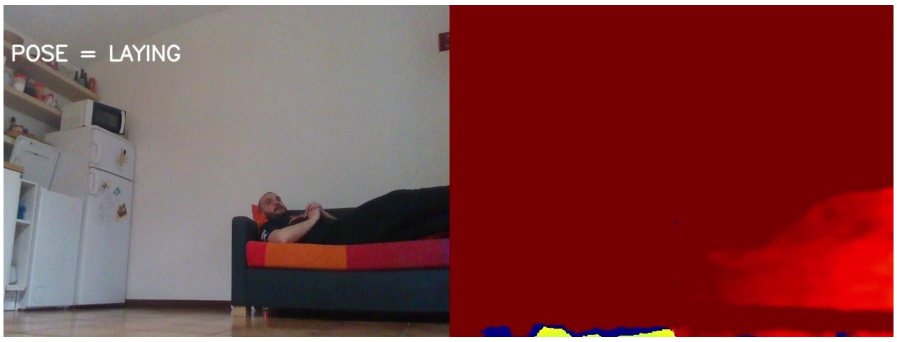
(b) Wrong Standing pose recognized



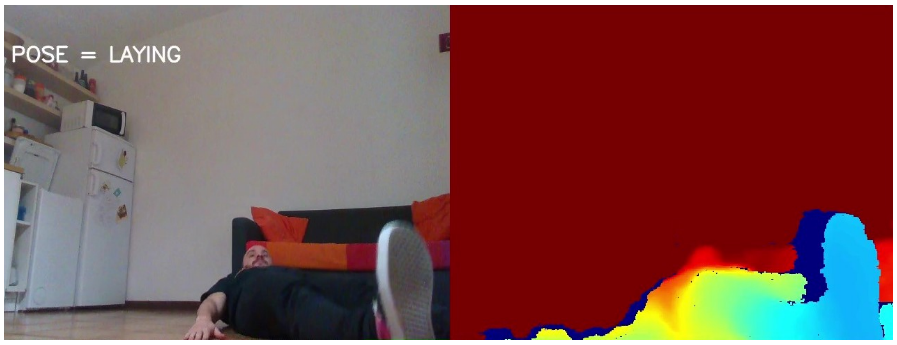
(c) Sitting pose on a couch



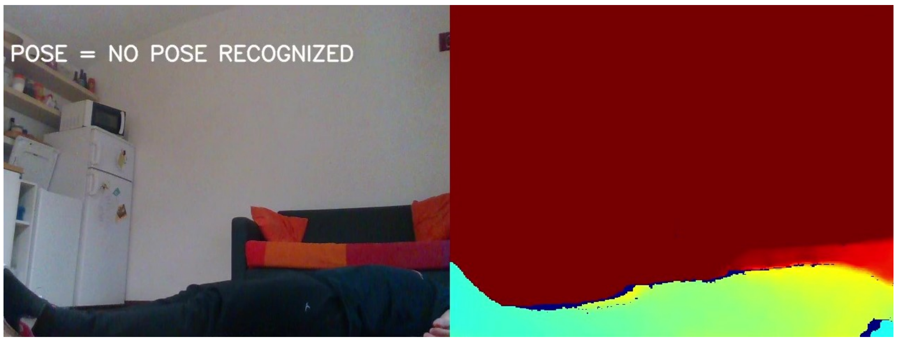
(d) Sitting pose on the ground



(e) lying pose on a couch



(f) lying pose on the ground



(g) NO pose recognized

Figure 5.4: Real time acquisitions of pose recognition algorithm.

Each of these figure is divided in two, the left part contains the RGB acquisition and the right one the Depth image. The colours in the depth image refer to the different distance that the object has from the camera. As can be seen, the background, that is approximately at a distance equal to three meters, has a red colour while the foreground has brighter colours as yellow and cyan.
In figure 5.4b the person is too far from the camera and the algorithm is not able to classify in a correct way the pose detected.
Figure 5.4g, where the head keypoints are not detected, no pose is classified by the algorithm, this because the leak of some keypoints can lead to a miss or a null pose classification.

Good results are obtained in the other cases and there is a performance improvement with respect to the previous model. It can be said that the addition of the dept information gives an improvement to the pose classification algorithm.

5.2 CNN

The CNN model has been the last developed model and a different approach is used.
The time to train the CNN model is very high. This is caused by the memory occupation of the dataset, hundreds of gigabytes. To overcome this problem the dataset needs to be loaded using the TensorFlow function $tf.data.Dataset.from\_generator()$.
The *tf.data.Dataset* API is used to create the input pipelines. This API helps in the creation of an input dataset and its important feature is that the full dataset not need to fit into memory. This is possible because the dataset is created iterating over the given input data in a streaming fashion.
Only one test has be performed on this model and the obtained results are reported below.

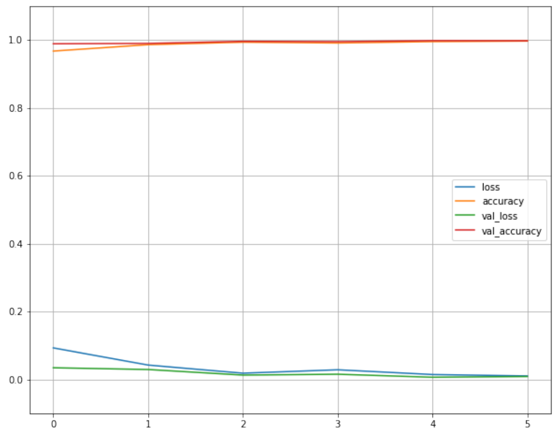
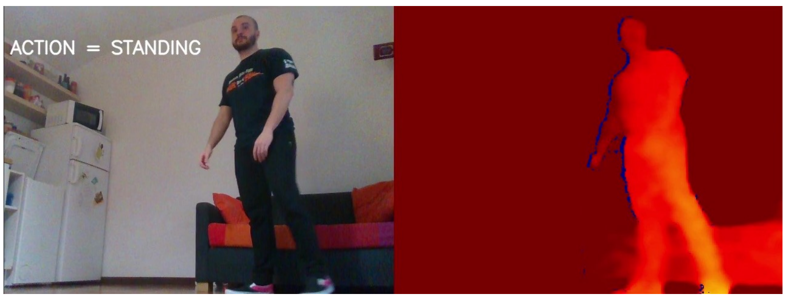


Figure 5.5: Accuracy and loss behaviour of CNN model.

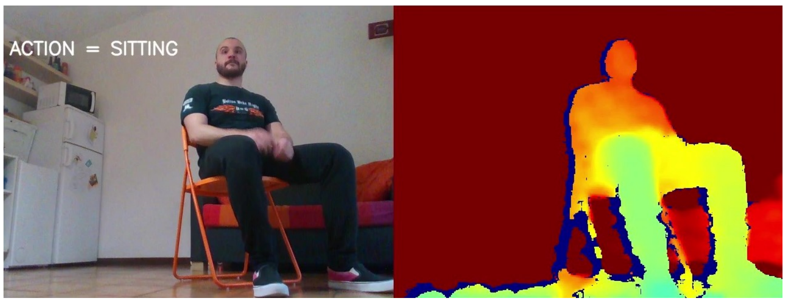
Figure 5.5 report the accuracy and loss trend over the training epochs, that in this case are 6.

The obtained accuracy, after the evaluation of the performance on the Test set, is **0.9974**.

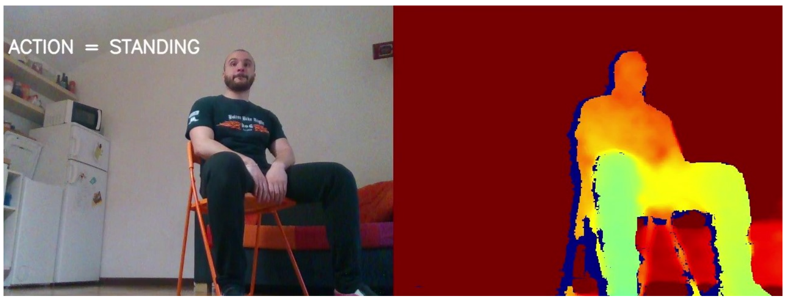
Also for this model the real time pose recognition algorithm is run and some images are acquired. 5.6



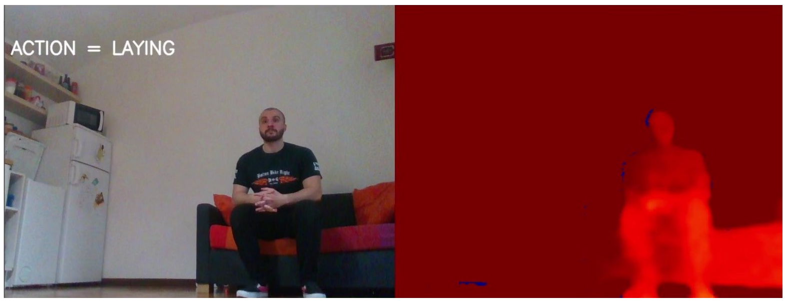
(a) Correct Standing pose recognized



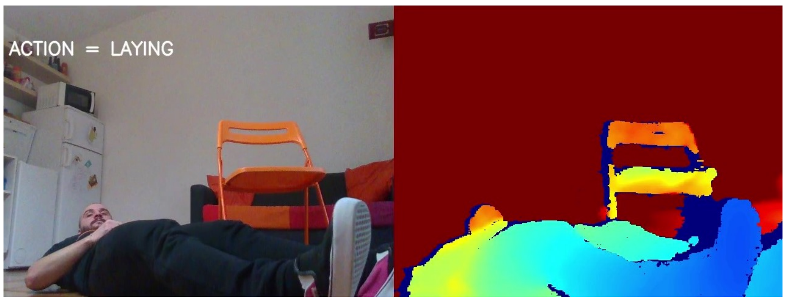
(b) Correct Sitting pose recognized



(c) Wrong Standing pose on a chair



(d) Wrong Sitting pose recognized



(e) Correct lying pose recognized

Figure 5.6: Real time acquisitions of pose recognition algorithm.

Despite the accuracy of this model is around the 99% the real time performances are poor.
If some keypoints are not detected the algorithm fails the pose classification. An example is reported in figure 5.6c, the feet keypoints are not detected and the pose detected is Standing instead of Sitting.
Despite the overall model performances are acceptable it is not reliable and the cause can be its complexity.

5.3 Conclusions and Future works

The obtained results satisfy the requests of the thesis and are considered the first steps of a broad project in which many other service tasks will be integrated.
The model that better fulfill the project requests is the MLP model using the dataset containing the depth information about the human keypoints. The obtained accuracy is high, the real time tests provide good results and the model is reliable.

Future works to improve this thesis project can be:

- add an object detector to better understand where the person is located, or what action is doing, inside the home environment recognizing a specific object in the room;
- add a fall detector algorithm to distinguish the dangerous situation from the normal one, e.g. a person sleeping in the bed or a person lying on the floor of the kitchen;
- implement the whole system on a wheeled robot to track an follow the person in the house environment;
- add a real action recognition algorithm.

Bibliography

[1] Yu Kong and Yun Fu. *Human Action Recognition and Prediction: A Survey.* 2018. arXiv: `1806.11230 [cs.CV]` (cit. on p. 3).

[2] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields.* 2019. arXiv: `1812.08008 [cs.CV]` (cit. on p. 4).

[3] URL: `https://github.com/google-coral/project-posenet` (cit. on p. 5).

[4] Alethea Carampel, Heidi Castillo, Maria Franchesca, T Riza, Theresa Batista-Navarro, and Prospero Naval. «TRACE: Tracking and Pose Recognition of Adults in Health Care Environments». In: (Dec. 2010) (cit. on pp. 6, 7).

[5] Y. KAMEDA. «A human motion estimation method using 3-successive video frames». In: *Proc. Conf. on Virtual Systems and Multimedia, Gifu, 1996* (1996). URL: `https://ci.nii.ac.jp/naid/10024346616/en/` (cit. on p. 6).

[6] M. D. Solbach and J. K. Tsotsos. «Vision-Based Fallen Person Detection for the Elderly». In: *2017 IEEE International Conference on Computer Vision Workshops (ICCVW).* 2017, pp. 1433–1442. DOI: `10.1109/ICCVW.2017.170` (cit. on pp. 7–9).

[7] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. *Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields.* 2017. arXiv: `1611.08050 [cs.CV]` (cit. on p. 7).

[8] Aurlien Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* 1st. O'Reilly Media, Inc., 2017. ISBN: 1491962291 (cit. on pp. 10, 15–18, 26).

[9] Thomas M. Mitchell. *Machine Learning.* 1st ed. USA: McGraw-Hill, Inc., 1997. ISBN: 0070428077 (cit. on p. 11).

[10] Intel. *Intel RealSense D400 Series Datasheet.* June 2020 (cit. on p. 47).

[11] *Intel RelaSense Depth Camera d435i.* URL: `https://www.intelrealsense.com/depth-camera-d435i/` (cit. on p. 47).

[12] *Google Coral TPU Device*. URL: `https://coral.ai/products/accelerator#description` (cit. on p. 48).

[13] *Keras*. URL: `https://keras.io/about/` (cit. on p. 49).

Acknowledgements

Il mio lavoro è giunto al termine anche se sono pronto per una nuova ripartenza...
Intanto per questo traguardo raggiunto

Ringrazio il mio relatore prof. Marcello Chiaberge,
la ringrazio per avermi dato la possibilità di svolgere questo progetto di tesi che mi ha permesso di scoprire e farmi appassionare al mondo del Machine Learning.

Ringrazio i miei tutor Vittorio e Francesco,
vi ringrazio per il supporto e la disponibilità che mi avete riservato durante lo sviluppo di questo progetto. I vostri consigli mi hanno guidato e fatto comprendere al meglio tutti i dettagli di questo percorso.

Ringrazio Mamma e Papà,
vi ringrazio per tutti i sacrifici che in questi anni avete fatto per me.
Per avermi insegnato ad avere fiducia in me stesso e credere in quello che faccio.
Per aver superato la distanza e il tempo che ho passato lontano da casa, perchè so che è un piacere avermi con voi.
Per essermi sempre stati vicino per qualsiasi esigenza e per avermi accompagnato lungo questo percorso che sarà un trampolino verso nuove avventure.
Ringrazio Serena,
ti ringrazio perchè sei sempre con me e ogni tanto butti giù un occhio per controllare il tuo fratellino.
Grazie perchè la persona che sono diventato è anche merito vostro.

Ringrazio i miei familiari per il sostegno e l'icoraggiamento che mi avete fornito in questi anni.
Ringrazio sopratutto Nonno Italo, i nonni Maddalena e Claudio, gli zii e cugini Molisani e Piemontesi per essersi sempre interessati e presi cura di me.

Ringrazio Irene, Pardini, Paddi e le tue altre mille personalità,
ti ringrazio per essere stata un'ottima compagna di studio, per aver condiviso momenti belli, brutti, pieni di ansia e gioia e con il tempo essere diventata mia Amika. Mi ricorderò tutte le cavolate dette insieme e le risate fatte, sopratutto dopo le 6 di sera.

Ringrazio i "Pisciaturi",
vi ringrazio perchè da una serata passata insieme a Roma è nata un'amicizia vera e forte. Un'amicizia che è passata da passeggiate, con annesse pustoline viola, a serate sotto le stelle. Un grazie perchè so che su di voi potrò sempre contare.

Ringrazio i miei amici di "Su",
vi ringrazio perchè, nonostante i lunghi periodi passati a Torino, il nostro legame non sia mai cambiato. Mi siete sempre stati vicini e con voi ho passato dei momenti indimenticabili.

Ringrazio i miei amici "Spessi",
vi ringrazio perchè non credevo che la palestra potesse creare un gruppo così unito. Quando ancora si poteva, era un piacere vedervi, chiacchierare e allenarmi con voi. Grazie a voi ho raggiunto obiettivi che non mi sarei mai aspettato di raggiungere.

Infine un grazie ai miei amici di Torino e ai compagni di avventure del Poli con i quali ho condiviso davvero dei bei momenti felici e spensierati che rimarranno impressi nei miei ricordi.

Questo momento segna la fine del mio percorso di studi concluso con la Laurea Magistrale. Un traguardo raggiunto seguendo un percorso di alti e bassi, ma che mi ha dato la possibilità di creare un bagaglio di conoscenze ampio e solido.
Ormai la pacchia è finita e si entra a far parte del "mondo dei grandi", guardandomi alle spalle voglio solo sorridere e ricordare i momenti di felicità e di crescita che ho vissuto.